Computing Project Documentation

Name: Chun Yin Wong (Jamin)

Candidate Number: 3328

Centre Number: 10816

Table of Content

[Analysis of the problem 4](#_Toc125983787)

[Problem Identification 4](#_Toc125983788)

[End users 5](#_Toc125983789)

[Research on existing games 5](#_Toc125983790)

[Super Mario Bros 5](#_Toc125983791)

[Syobon Action (Cat Mario) 6](#_Toc125983792)

[Game Analysis 6](#_Toc125983793)

[Current System 7](#_Toc125983794)

[Investigation 8](#_Toc125983795)

[Questionnaire 1 (Features of the Game) 8](#_Toc125983796)

[Result of Questionnaire 1 9](#_Toc125983797)

[Questionnaire 2 (Detailed Specification) 10](#_Toc125983798)

[Result of Questionnaire 2 11](#_Toc125983799)

[My Approach 12](#_Toc125983800)

[Limitation 12](#_Toc125983801)

[Solution Requirements 13](#_Toc125983802)

[Hardware 13](#_Toc125983803)

[Software 13](#_Toc125983804)

[Success Criteria 14](#_Toc125983805)

[Design of the solution 16](#_Toc125983806)

[Problem decomposition 16](#_Toc125983807)

[Initial User Interface Design 19](#_Toc125983808)

[Main Screen 19](#_Toc125983809)

[Menu 20](#_Toc125983810)

[Ranking Board 21](#_Toc125983811)

[Instructions Screen 22](#_Toc125983812)

[In-game menu 22](#_Toc125983813)

[Gameplay Screen 23](#_Toc125983814)

[Win Screen 24](#_Toc125983815)

[Final Design 25](#_Toc125983816)

[Main screen 25](#_Toc125983817)

[Rank Board 26](#_Toc125983818)

[Gameplay screen 27](#_Toc125983819)

[Class Diagram 29](#_Toc125983820)

[Key Data Structure Design 36](#_Toc125983821)

[Main 36](#_Toc125983822)

[Game 36](#_Toc125983823)

[Player 36](#_Toc125983824)

[Map 37](#_Toc125983825)

[Algorithms 37](#_Toc125983826)

[Game 37](#_Toc125983827)

[Player 38](#_Toc125983828)

[Map 40](#_Toc125983829)

[Testing Data 40](#_Toc125983830)

[Developing the Coded Solution 42](#_Toc125983831)

[Iteration 1 42](#_Toc125983832)

[Iteration 1 Testing 47](#_Toc125983833)

[Feedback from end user 47](#_Toc125983834)

[Iteration 2 48](#_Toc125983835)

[Iteration 2 Testing 51](#_Toc125983836)

[Feedback from end user 51](#_Toc125983837)

[Iteration 3 53](#_Toc125983838)

[Iteratioh 3 Testing 57](#_Toc125983839)

[Feedback for iteration 3 58](#_Toc125983840)

[Iteration 4 59](#_Toc125983841)

[Iteration 4 Testing 65](#_Toc125983842)

[User feedback 66](#_Toc125983843)

[Iteration 5 67](#_Toc125983844)

[Iteration 5 testing 77](#_Toc125983845)

[Iteration 6 79](#_Toc125983846)

[Iteration 6 Testing 82](#_Toc125983847)

[Testing results 84](#_Toc125983848)

[Result of testing 86](#_Toc125983849)

[User Testing 87](#_Toc125983850)

[Final version of the code 89](#_Toc125983851)

[Evaluation 138](#_Toc125983852)

# Analysis of the problem

## Problem Identification

The current games in the market have a lot of elements and detailed graphics, allowing the user to immerse in the virtual environment. However, most of these games contain a lot of different mechanics, like different skill combinations, tactics and characters. The well-known classic Super Mario Bros was created more than 30 years ago. It is no longer compatible with modern machines, and the game’s visual quality is poor in the current standard. It will inherit a similar idea of a horizontal platform game.

This games often take hundreds of hours of playing and reading to learn and master these games. Hence, I have used the classic game Super Mario Bros as an inspiration and created a platform game which only has the necessary mechanics to minimise the learning curve of this game.

I will create a platform game using Python and Pygame library. It will have a similar layout as the Super Mario Bros. Super Mario Bros is a platform game developed by Nintendo and first released in 1985. The player control Mario as they travel the Mushroom Kingdom to rescue Princess Toadstool.

Similarly, in this game, the user will control a character. The objective is to pass through a series of obstacles and enemies. The game ends when the player reaches the flag pole a the end of each level or the player decides to quit.

It will have two levels with different maps and layouts. The map will be predetermined and imported locally. There will be a death-count counting the number of death the player has in that level. The number of enemies in this game is reduced, but a large variety of traps are introduced to increase the dynamic of the game.

## End users

The users of the game are a group of teenage students ranging from the age of 16 to 25. These users have a shorter attention span. This game is straightforward, and the times needed for each level are relatively short, which suits the short attention span and busy life of the current adolescents. Although the basic idea and concept of the platform game are similar to the original Super Mario Bros, it has a different play style as the player needs to focus on different traps instead of different enemies. It can give a new taste of a similar game to users that have tried the Super Mario series.

## Research on existing games

### Super Mario Bros

It is a platform game. The character Mario is controlled by the player, who has to go through the Mushroom Kingdom. At each level, there are coins throughout the map that can be collected. There is also special gear that Mario can be collected, which will then give him special powers like invincibility, and shooting fireballs, et al. There are different enemies, including Goombas, who can be stepped on to defeat; Koopas, who are turtle-like creatures that can be bounced around; Browser, the boss of the game.

![Graphical user interface

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEASABIAAD/4RCGRXhpZgAATU0AKgAAAAgAA4KYAAIAAAAfAAAIPodpAAQAAAABAAAIXuocAAcAAAgMAAAAMgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEFydHJvc2VzdHVkaW8gfCBEcmVhbXN0aW1lLmNvbQAAAAHqHAAHAAAIDAAACHAAAAAAHOoAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAP/hCopodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpyaWdodHM+PHJkZjpBbHQgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpIHhtbDpsYW5nPSJ4LWRlZmF1bHQiPkFydHJvc2VzdHVkaW8gfCBEcmVhbXN0aW1lLmNvbTwvcmRmOmxpPjwvcmRmOkFsdD4NCgkJCTwvZGM6cmlnaHRzPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMAAwICAwICAwMDAwQDAwQFCAUFBAQFCgcHBggMCgwMCwoLCw0OEhANDhEOCwsQFhARExQVFRUMDxcYFhQYEhQVFP/bAEMBAwQEBQQFCQUFCRQNCw0UFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFP/AABEIA3cGQAMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/AOoooor+qD48KKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooq1DAnl+dcf6j/0OpcrALbwJ5fnXH7uBP/H6juJ3uJPn+5/BH/conuHuH+f7n8Ef9yoKzSe7AKKKK2AKKKKACiiigAooooAKKKKACiiigAooooAKKKkt7d55NiffpbAFvbvO+xPv1PcXCQR/Zbb/AFf8cn9+i4nSBPstt/q/45P79VKy/iasAooorYAooooAKKKKACiiigAooooAKKKKACiiigAooqeG3Sd9jv5b/wAFJu2oEFT29u9w/wDzzRPvySfwUR2jyTOj/u9n3/M/gqS4uE8vyYf3cH/odZXb0QxJ508vyYvkg/8AQ6q0UVqlyiCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFW7eBPL864/dwJ/4/RBAnl+dc/6j/0Oo57h7h/n+5/BH/crHWWiGE873Enz/c/gj/uVBRRWqVtEIKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABUlvbvcSbE+/Rb273EmxPv1NcXCQJ9mt/ufxyf36ybeyAJ50gT7Nb/AHP45P79VaKKtKwBRRRVAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAJto21Yt7dLiTY7+XJ/BRHaPJM6P+72ff8z+Cp9oh6hb273D/wDPNE+/JJ/BUlxcJ5fkw/u4P/Q6Li4Ty/Jh/dwf+h1UrNJy1YBRRRWwgooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKtQwJ5fnXH+o/wDQ6lysAtvAnl+dcfu4E/8AH6juJ3uJPn+5/BH/AHKJ7h7h/n+5/BH/AHKgrNJ7sAooorYAooooAKKKKACiiigAooooAKKKKACiiigAooqS3t3nk2J9+lsAW9u877E+/U9xcJBH9ltv9X/HJ/fouJ0gT7Lbf6v+OT+/VSsv4mrAKKKK2AKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKtQQJ5fnXH+o/wDQ6lvlALO3Ty/OuP8AUf8AodSapvkdJv8AWQP/AKny/wCD/YqpcXD3D73/AOARx/wVJaXnkb0dPMgf78dY2f8AEGQUVPPafZ/nR/Mgf7klQVummroQUUUUwCiiigAooooAKKKKACiiigAooooAKKKkt7d55NiffpbAFvbvPJsT79T3FwkEf2W2/wBX/HJ/fouJ0gT7Lbf6v+OT+/VSsv4mrAKKKK2AKKKKACiiigAooooAKKKKACiiigAooooAKKKKACp7e3e4f/nmiffkk/got7d7h/8Anmiffkk/gqS4uE8vyYf3cH/odYtvZAWpLxL+H7Mj+Xs+5JJ/H/v1lSRvHI6OnlulFXo5Ev08mV/LnT7kn/tN6n+FtsMo0USRvHI6OnlulFdAgooooAKKKKACiiigAooooAKKKKACiiigAooq1BAnl+dcf6j/ANDqW+UBbe3Ty/OuP3cH/odVKkuLh7iTe/8AwCP+5UdRTv1AKKKK1AKKKKACiiigAooooAKKKKACiiigAooooAKKKKACrdnbp5fnXP8AqP8A0OiCBPL865/1H/odV7i4e4fe/wDwCOP+CsXeWiGW9U3yOk3+sgf/AFHl/wAH+xVACrFneeRvR08yB/vx0XFn9n+dH8yB/uSUofu/cYiCiiitwCiiigAooooAKKKKACiiigAooooAKKKKACpLe3e4k2J9+i3t3uJNiffqa4uEgT7Nb/c/jk/v1k29kATzpAn2a3+5/HJ/fqrRRVpWAKKKKoAooooAKKKKACiiigAooooAKKKKACiiigAooqeCB7h/+eaJ9+ST+Ck3bVgFvbvcP/zzRPvySfwVekvEv4PsyP5ez7kkn8f+/VW4uE8vyYf3cH/odUy1YcntNWMWSN45HR08t0oq9HIl9H5Mr+XOn3JP/ZHqjJG8cjo6eW6VUZN6MQUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVbt7dPL864/dwf8AodEECeX51z/qP/Q6guLh7iTe/wDwCP8AuVjrLRDI6KKK2EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRVqCBPL864/1H/odS3ygFnbp5fnXH+o/wDQ6k1TfI6Tf6yB/wDU+X/B/sVUuLh7h97/APAI4/4KktLzyN6OnmQP9+OsbP8AiDIKKnntPs/zo/mQP9ySoK3TTV0IKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFSW9u88mxPv0tgC3t3nk2J9+p7i4SCP7Lbf6v+OT+/RcTpAn2W2/1f8cn9+qlZfxNWAUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVagt02edcf6j+CP+/Ut2AWCBPL865/1H/odQT3D3cm9/wDgEf8AconuHu5N7/8AAI/7lR1mk92AUUUVsBPaXnkb0dPMgf78dFxafZ/nR/Mgf7klQVPaXnkb0dPMgf78dYtNaoCCip57X7P86P5kD/ckqCtU01dAFFFFMAooooAKKKKACiiigAooqS3t3nk2J9+lsAW9u88mxPv1PcXCQR/Zbb/V/wAcn9+i4nSBPstt/q/45P79VKy/iasAooorYAooooAKKKKACiiigAooooAKKKKACiiigAooooAKnt7d7h/+eaJ9+ST+Ci3t3uH/AOeaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MXyQf+h1VoorVLlAKKKKYF6ORL9PJlfy50+5J/7TeqMkbxyOjp5bpRV6ORL9PJlfy50+5J/wC03rD+F6DKNFEkbxyOjp5bpRW4gooooAKKKKACiiigAooooAKKKtQW6bPOuP8AUfwR/wB+pbsAtvbps864/wBR/wA8/wC/UFxcPcSb3/4BH/couLh7iTe//AI/7lR1mk73YBRRRWwBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVbggTy/Ouf9R/6HRBbp5fnXH+o/gj/v1BPcPdyb3/4BH/crHWWiASe4e7fe/wDwCP8AuUyiitUraIAqe1u/I3o6eZA/346gooaUlZgT3Fp9n+dH8yB/uSVBU9peeRvR08yB/vx0XFp9n+dH8yB/uSVkm07MCCiiitgCiiigAooooAKKKKACiiigAqS3t3uJNiffot7d7iTYn36muLhIE+zW/wBz+OT+/WTb2QBPOkCfZrf7n8cn9+qtFFWlYAoooqgCiiigAooooAKKKKACiiigAooooAKKKKACiip4IXnf/nmiffkk/gpN23ALe3e4f/nmiffkk/gqS4uE8vyYf3cH/odFxcJ5fkw/u4P/AEOqlZJN6sYUUUVsIKuxyJfp5Mr7J0+4/wD7I9UqKmUeYAkjeOR0dPLdKKvRyJfJ5Mr+XOn3JP8A2m9UZI3jkdHTy3Ss4yb0YBRRRWwBRRRQAUUUUAFFFFABRRRQAVbt7dNnnXH+o/55/wB+iC3Ty/OuP9R/BH/fqC4uHuJN7/8AAI/7lY6y0QCXE7zvvf8A4BH/AHKZRRWqVtEAUUUUwCiiigAooooAKKKKACiiigAooooAKKKKACiirUFumzzrj/UfwR/36luwCwQJ5fnXP+o/9DqCe4e7k3v/AMAj/uUT3D3cm9/+AR/3KjrNJ7sAooorYCe0vPI3o6eZA/346Li0+z/Oj+ZA/wBySoKntLzyN6OnmQP9+OsWmtUBBRU89r9n+dH8yB/uSVBWqaaugCiiimAUUUUAFFFFABRRRQAUUVJb27zybE+/S2ALe3eeTYn36nuLhII/stt/q/45P79FxOkCfZbb/V/xyf36qVl/E1YBRRRWwBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRVqC3TZ51x/qP4I/79S3YBYLdPL864/1H8Ef9+oLi4e4k3v8A8Aj/ALlFxcPcSb3/AOAR/wByo6zSe7AKKKK2AKKKKACiiigCe0vPI3o6eZA/346Li0+z/Oj+ZA/3JKgqe0vPI3o6eZA/346xaa1QEFFTz2v2f50fzIH+5JUFappq6AKKKKYBRRRQAUUVJb27zybE+/S2ALe3eeTYn36nuLhII/stt/q/45P79FxOkCfZbb/V/wAcn9+qlZfxNWAUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFT29u9w/8AzzRPvySfwUW9u9w//PNE+/JJ/BUlxcJ5fkw/u4P/AEOsW3sgEnnTy/Ji+SD/ANDqrRRWqXKAUUUUwCiiigAooooAvRyJfp5Mr+XOn3JP/ab1RkjeOR0dPLdKKvRyJfp5Mr+XOn3JP/ab1h/C9BlGiiSN45HR08t0orcQUUUUAFFFFABRRVqCFI0+03H+o/gT+/Ut2ALe3SOPzrj/AFH8Ef8AfqG4uHuJN7/9+/7lFxcPcSb3/wD2KjrNJ3uwCiiitgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAq3Bbp5fnXH+o/gj/AL9EFunl+dcf6j+CP+/UFxcPcSb3/wCAR/3Kx1logEuJ3nfe/wDwCP8AuUyiitUraIAooopgFFFFABU9rd+RvR08yB/vx1BRSaUlZgT3Fp9n+dH8yB/uSVBU9peeRvR08yB/vx0XFp9n+dH8yB/uSVkm07MCCiiitgCiiigAooooAKkt7d7iTYn36Le3e4k2J9+pri4SBPs1v9z+OT+/WTb2QBPOkCfZrf7n8cn9+qtFFWlYAoooqgCiiigAooooAKKKKACiiigAoqTy/wDbT/vtKj8v/pon/fcdYe2h3HZhRRRW4gooqeCF53/55on35JP4KTdtwC3t3uH/AOeaJ9+ST+CpLi4Ty/Jh/dwf+h0XFwnl+TD+7g/9DqpWSTerGFFFFbCCiiigAooooAKuxyJfp5Mr7J0+4/8A7I9UqKmUeYAkjeOR0dPLdKKvRyJfJ5Mr+XOn3JP/AGm9UZI3jkdHTy3Ss4yb0YBRRRWwBRRRQAUUUUAFWre3SOPzrj/UfwR/36W3gSNPtNx/qP4I/wC/UFxcPcSb3/8A2Kwu5aIBJ53nfe//AH7/ALlMoorZK2gBRRRTAKKKKACiiigAooooAKKKKACiiigAooooAKKKtQW6bPOuP9R/BH/fqW7ALBbp5fnXH+o/gj/v1BcXD3Em9/8AgEf9yi4uHuJN7/8AAI/7lR1mk92AUUUVsAUUUUAFFFFAE9peeRvR08yB/vx0XFp9n+dH8yB/uSVBU9peeRvR08yB/vx1i01qgIKKnntfs/zo/mQP9ySoK1TTV0AUUUUwCiiigAooqS3t3nk2J9+lsAW9u88mxPv1PcXCQR/Zbb/V/wAcn9+i4nSBPstt/q/45P79VKy/iasAooorYAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooq1BCkaedcf6j+CP+/Ut2AWC3Ty/OuP8AUfwR/wB+oLi4e4k3v/37/uUXFw9xJvf/AL9/3KjrNJ7sAooorYAooooAKKKKACiiigAooooAntLzyN6OnmQP9+Oi4tPs/wA6P5kD/ckqCp7S88jejp5kD/fjrFprVAQUVPPa/Z/nR/Mgf7klQVqmnqgCiipLe3eeTYn36NgC3t3nk2J9+p7i4SCP7Lbf6v8Ajk/v0XE6QJ9ltv8AV/xyf36qVl/E1YBRRRWwBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVPb273D/8APNE+/JJ/BRb273D/APPNE+/JJ/BUlxcJ5fkw/u4P/Q6xbeyASedPL8mL5IP/AEOqtFFapcoBRRRTAKKKKACiiigAooooAKKKKAL0ciX6eTK/lzp9yT/2m9UZI3jkdHTy3Sir0ciX6eTK/lzp9yT/ANpvWH8L0GUaKJI3jkdHTy3SitxBRRVqCBI0+03H3P4E/v1LlYAt7dI4/tNx9z+CP+/UNxcPcSb3/wD2KLi4e4fe9R1CTvdgFFFFagFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABVuC3Ty/OuP8AUfwR/wB+kt7dI4/OuP8AUfwR/wB+obi4e4k3v/37/uVjdy0QCTzvO+9/+/f9ymUUVqlbQAooopgFFFFABRRRQAUUUUAFT2t35G9HTzIH+/HUFFJpSVmBPcWn2f50fzIH+5JUFT2l55G9HTzIH+/HRcWn2f50fzIH+5JWSbTswIKKKK2AKkt7d7iTYn36Le3e4k2J9+pri4SBPs1v9z+OT+/WTb2QBPOkCfZrf7n8cn9+qtFFWlYAoooqgCiiigAooooAKKKKACiiigBI43k/1aPJ/wBc68N8ReLofj78TNa/Z8XTH0C8v7qSzPiGWYzxp9l/0ri18tHy/kbP9Z/HUP7ZWqXui/B/T7nTr650+4/t6BBJazvG+zyLr+5Xtfw10jTrX9kLwp4ngsLKPxPJosE766lqiX+97vY7/atm/wD1fyffr8D8SOMquR0fqeHXvz93m/xR0O2PJh6E8VP7J8m+PP2CT4I8SSaRJ45ju2SGGbzBpTxjD/8AA655f2NSrZ/4TCM/9w9//i6+mLi8nvLjz7m6mvJ/9X5lw7ySf+P1BX80U+J81Ubzr/8AksT88q8SZhKpOdOfuf8AbonwR/aBj+POr6/bW3ha60ZtOhS6k8y++1798nl/8802V6hnmvjz9pyUeFPDfh640NU0KWa4uo5pNLRbUum2P5H8sJvr6N+DvjGDxN4V8L6en2r7Umg2Ml1d3f8Aq/kgj8yTzPMr+weC+KJZ9guetH3kfa4etTxdCGKpw5bnfW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDonuE8vyYv3cH/AKHVSv0xJvVlBRRRWwgooooAKKKKACiiigAooooAKuxyJfp5Mr7J0+4//sj1SoqZR5gCSN45HR08t0oq9HIl8nkyv5c6fck/9pvVGSN45HR08t0rOMm9GAUUUVsAVbt7dI0+1XP3P4I/79FvbpHH9pufufwR/wB+oLi4e4fe9Y/xNEAXE7zyb3//AGKjoorVK2gBRRRTAKKKKACiiigAooooAKKKKACiiigAooooAKKKtQQpGnnXH+o/gj/v1LdgFgt08vzrj/UfwR/36guLh7iTe/8A37/uUXFw9xJvf/v3/cqOs0nuwCiiitgCiiigAooooAKKKKACiiigCe0vPI3o6eZA/wB+Oi4tPs/zo/mQP9ySoKntLzyN6OnmQP8AfjrFprVAQUVPPa/Z/nR/Mgf7klQVqmnqgCiipLe3eeTYn36NgC3t3nk2J9+p7i4SCP7Lbf6v+OT+/RcTpAn2W2/1f8cn9+qlZfxNWAUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFT29u9w/wDzzRPvySfwUW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDrFt7ICpRRRWwBRRRQAUUUUAFFFFABRRV/T9P+0fPN/q/4I/8AntWU58gEdvAkafabj/UfwR/36guLh7iTe/8A+xRcXD3Em9/+/f8AcqOkk92AUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBPZ3n2fejp5kD/fjouLP7P86P5kD/ckqCrelyOZPJ2eZG/346wenvoCC3t3nfYn36nuLhII/stt/q/45P79T38aWcGy2fzIH/5b/wB+s2hfvNQCiiitwCiiigAooooAKKKKACiiigAooooAKKKKACiiigBDVi3t3nb/AJ5on35JP4KW3t3uJP8AnnGn35JP4Kmv/wDR9kMP/Hr99JP+e3+3WDnryIZHcTp5fkxfu4P/AEOqtFFbJcogooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAXo5Ev08mV/LnT7kn/tN6oyRvHI6OnlulFatnGlxDG939/8A5Y+Y/wB+uf8AhDKlvbpHH9puPufwR/36huLh7h971JeSTSTSed/rP+ef9yq9VT7sQUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFW7eBI0+03H+o/gj/v0+w0/7R883+r/gT/ntVW4uHuJN7/8Afv8AuVhf2jshhcTvPJvf/wDYqOiitkraCCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABU9rd+RvR08yB/vx1BRSaUlZgT3Fn5Hzo/mQP8AckplvbvO+xPv1Ppcj+Z5OzzEf78dT6hGlnBstn8yB/8Alv8A36w53fk6jK886QJ9mt/ufxyf36q0UVtFcogoooqgCiiigAooooAKKKKACiiigAooooDfY+e/24Pm+C+nen/CRWv/AKT3ddF8K/GuuyfAvwpoT6rM2lDTVj+x7E8vZ9okf+5XdfEL4daD8UfD8ejeIYbmewhuo7tI7SbyH8xEdP8A2d68Js7/AMVeB/ijN4YfSbjT/hfptxPZ2usahZBIFiKv5G+8ePZ9/Z89fzV4oZDicf8A7XTh7kf/AG2Jx5pTrYrBeyw89YfEenUVh/8ACc+GP+ho0P8A8GUP/wAXVmx8YeGbmRg3irQYUVN7OdTh2J/4/X8yLA4l/wDLpn5n9RxO3s2eQ/tXac+o+GvDCI6oi3N07TsfkRNkdd14T8QT6Z4T8PJol1NZwf2Xap58f+sm/cR/frgt3iD4sLJo3jvS7qx0exVLrT5ILUWJuXchHkf93l/k/h/grtbOzTT7G0sof9RawRwJ5n9xI/Ljr+u/D/JsTlWCvif694/WcHh54LBU8LP4z2Lwf4zstQsdNsrnUXuNadH3+Yj/ADyfvP8Alp9z7ldbXivw/wD+R00n/fk/9ESV7VX7ZRlzxLmFFFFdJmFFFFABRRRQAUUUUAFFFFABRRRQAVdjkS+TyZX8udPuP/7I9UqKmUeYBJI3jkdHTy3SrdvbpHH9pufufwR/36uWcaXEMb3f+s/5YeY/36oXkk0k0nnf6z/nn/crHn9p7gxlxcPcPveo6KK3StohBRRRTAKKKKACiiigAooooAKKKKACiiigAooooAKKKv6fp/2j55v9X/BH/wA9qynPkAjt4EjT7Tcf6j+CP+/UFxcPcSb3/wD2KLi4e4k3v/37/uVHSSe7AKKKK2AKKKKACiiigAooooAKKKKACiiigAooooAns7z7PvR08yB/vx0XFn9n+dH8yB/uSVBVvS5HMnk7PMjf78dYPT30BBb27zvsT79T3FwkEf2W2/1f8cn9+p7+NLODZbP5kD/8t/79ZtC/eagFFFFbgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACGrFvbvO3/ADzRPvySfwUtvbvcSf8APONPvySfwVNf/wCj7IYf+PX76Sf89v8AbrBz15EMjuJ08vyYv3cH/odVaKK2S5RBRRRTAKKKKACiiigAooq1BAkafabj7n8Cf36lvlALe3SOP7Tcfc/gj/v1DcXD3Em9/wD9ii4uHuH3vUdQk73YF7/kJf8AX9/6O/8As6o0Ve/5CX3/APj6/wCen/Pb/wCzqf4foBRoo/1dFbgFFFFABRRRQAUUUUAFFFFABRRRQAUVJb27zzRwwp5ju+xK6O88E/ZNLu5/tvmXUKb3SNK5amIp02lM05DmKtSTpHD5MX8f33/v1VorosmZk9peeRvR08yB/vx0XFp9n+dH8yB/uSVBU9peeRvR08yB/vx1m01qgIKKnurTyNjo/mQP9ySoK1TTV0AUUUUwCiiigAooooAKKKKACiiigAooooAKnt7d7h/+eaJ9+ST+Ci3t3uH/AOeaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAXFwnl+TD+7g/9Dot7hNnkzfvIH/8cqpRR7NAT3Fu9vJ/z0T+CT+/UFW7e4TZ5M37yB//AByoLi3e3k/56R/wSR/x0JvZgR0UUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRU1p5Me95v3mz7kf9+k3ZASW8CQJ9puP9X/BH/fqG4uHuH3vRcXD3Em9/v1HWaTvdgXo5Ev08mV/LnT7kn/sj1RkjeOR0dPLdKKtyXCXFrsm/wBen3JKm3s9hlSiiitxBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABVu3t0jT7Vc/c/gj/v0W9ukcf2m5+5/BH/fqC4uHuH3vWP8AE0QBcXD3Em9//wBirX/IS/6/f/R3/wBnVGih0+wBRV7/AJCX3/8Aj6/56f8APb/7OqP+rpptgFFFFagFFFFABRRRQAUUUUAFFFFABRRUlvbvPNHDEnmO7+WlJu2rAjorqLzwR9j0u6n+2+ZdwpvdI0rljXPRrwrfAaezLklwkcHkw/x/fk/v1HaXn2fejp5kD/fjqCitPZozJ7i0+z/Oj+ZA/wBySoKntLzyN6OnmQP9+Oi7s/s+x0fzIH+5JSTadmBBRRRWwBRRRQAUUUUAekfBH4T6f8WbjXkvtTvbA2CQOn2Ty/n3+Z9/zEf+5Xrn/DHOgt18Sa5j/t1/+MV8pXlmt5C6SokgdNn7xK+vPgb8bh441CDwqmiNp40/TBJ9rNzv3iPy0wE2D+/X5txFHNMJJ4vDVn7Pt/KelhfY1PcmUv8AhjnQP+hk1z/yV/8AjFH/AAxzoP8A0Mmufla//GK+guaOa/Pv7ezP/n+z0/q1H+Q+e/8AhjrQduf+Ek1z8rX/AOMVD/wyD4ezg+Kdaz9bX/43Vz49ftHH4M6xp+kjQJNW+328k/nLeJFs2OFI5X3r5I/Z+/Z+t/jm+s21vrdrpJ0hLUPIbD7V52/f/tps/wBX/wCP187ieN8yhiPquFqOc/Xl/wDbT5nGZnRo4mGCw1Hnn/4CfVLfsh6Bu48Ua0fbNtn/ANF1g/ED9g/wn8SPBupeGL/xXr8drf8AlmR7drbzPkkSQf8ALP8A6Z185fDn4Bp8Rfin4g8ERX9rYTaSb7N81h53m+RcpD9zemzfv38PU3h+3tv2ff2hHCpDrEvhmeZZHTZa+cXtQm/+PZ/r68mpx1m06S+txfJL3d1/8iefDO3TcK1TC8kObl5uY1da/wCCNfwt0nSLy8PjfxiRbxPJgta84Gf+eFfMPx5/Yy8JfDXUNC0bRfEmuTWN5aSXU32yCEuzpJs/gr9XPCXxOX4ufBfV/EEdjJp6PHeQeTJIHP7sunUfSvhv9rj/AJGrwn/2C5//AEfWmTYp4rijB4R/wZwlL/yWR9jUrqpQ9tRPILfVEk8yC8/eWkz+Z/00gk/56R1UvNPfT5Nj/vEf94kkf+reOoK6XwXp6eJL6TR5rr7PB5Ek6SbPM2Sf9M6/p5Lojx/Uj+H3/I6aT/vyf+i5K9qqvo+kTeG9DsdLmmS48hP+PiP5I3+erFevQjyQM5hRRRXSZhRRRQAUUUUAFFFFABRRRQAUUUUAFWreBIE+03H+r/gj/v0yz8mPzHm/ebPuR/36ZcXD3Em9/v1i7y0ALi4e4fe9Wo5Ev08mV/LnT7kn/sj1RoodMAkjeOR0dPLdKKtSTpPbbJf9en3JKq1pFvqAUUUVQBRRRQAUUUUAFFFFABRRRQAUUUUAFFFWoIEjT7Tcfc/gT+/Ut8oBb26Rx/abj7n8Ef8AfqG4uHuJN7//ALFFxcPcPveo6hJ3uwL3/IS/6/v/AEd/9nVGir3/ACEvv/8AH1/z0/57f/Z1P8P0Ao0Uf6uitwCiiigAooooAKKKKACiiigAooooAKKkt7d55o4YU8x3fYldHeeCfsml3c/23zLqFN7pGlctTEU6bSmachzFWpJ0jh8mL+P77/36q0V0WTMye0vPI3o6eZA/346Li0+z/Oj+ZA/3JKgqe0vPI3o6eZA/346zaa1QEFFTz2v2f50fzIH+5JUFaKSYBRRRVAFFFFABRRRQAUUUUAFFFFABRRRQAVPb273D/wDPNE+/JJ/BRb273D/880T78kn8FSXFwnl+TD+7g/8AQ6xbeyALi4Ty/Jh/dwf+h0W9wmzyZv3kD/8AjlVKKPZoCe4t3t5P+eifwSf36gq3b3CbPJm/eQP/AOOVBcW728n/AD0j/gkj/joTezAjooorYAooooAKKKtQQJGn2m4+5/An9+pb5QC3t0jj+03H3P4I/wC/UNxcPcPvei4uHuH3vUdQk92AUUUVqAUUUUAXo/8AiZx7H/4+v4JP79UZI/Lk2PRV6ORNTj2P+7uv4JP79Yfw/QZRookjeOTY/wB+itxBRRRQAUUUUAFFdJZ6HbWfmJcWyXd2kfmTySXXkQQ/9M9/9+qGsaOlvD50KPb7PneDfvj2f30euCGMhUnyI09mZVHmVJb273k6QxI8jv8AwR11tvb/APCJeHfts1l5l877H3/wVdfEKlotx04HOeH7xNP1u1un/wBWj/PXSX6ahpt1rOyym1CDU0+R7f8Agqp9ns/GVrI9siWmqw/fj/v1k2mv6to8clskzx7Pk8uRN+yvNqQeIndfGjT+GSap4cfRNPtZrmZPtc3/ACwrJHFS3F5PdzST3Mz3E7/8tJKiHNevQVRQ/ebnOLRRRW4gooooAKKKKACiiigAooooAKKKKACiiigAqe3t3uH/AOeaJ9+ST+Ci3t3uH/55on35JP4KkuLhPL8mH93B/wCh1i29kAk86eX5MXyQf+h1VoorVLlAKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVbt7dI0+1XP3P4I/79FvbpHH9pufufwR/36guLh7h971j/E0QBcXD3D73qOiitVpogCiiimAVdj/4mUex/wDj6/gk/v1SoqZRuASR+XJseir0cialHsf93dfwSf36oyRvHJsf79QpX0AKKKK1AKKKKAE3UtJiuks9DtrTzEuLZLu6RPMmkkuvIgh/6Z+Z/frkr4iFLc0hT9oc3nFG6tbWNHS3g8+FHt9nzvBv3x7P+eiPWdb273lwkMKPJI//ACzjrSnXhOHORyEfmZq9oN4mn61a3TfcR/nrpILb/hEPDv22ay8y+d9j7/4KqfZ7PxnayPbIlpqsP34/79edPFRqXVvcOjkLeoR6hpl1rGyym1CDU0+R4P4KwtU8Ovomn2s1zMn2ub/lhTLPX9W0mN7ZJnj2fJ5cieZsqjcXk93NJNczPcTv/wAtJKWHoVac9HoE6hHRRRXsnMFFFFABRRRQAUUUUAFFFFABU1jqV5pM3nWN5eWE7ps8y0meB9n9zelQ0VEoxmuWS0Gj6m+Bvxw03UrPwx4NuP7Vu/ED2vkSX90u9JJEjd3+cvv/AID+lfQHT8q/OLSdX1DQdQgv9MvHsb6Hfsmh++n7vZX038DPjppsnh7TtI8TeIbi88T3l68KLPbO27fITCm9I/L+5ivxriLhyeHbxWF1T+euv/kp7mGxX2Jnkf7cxX/hZHg/YMf6Hz9ftUde+/Ej4peDf2eI9PludBmD6wzIraPaxZOzvJl0/v8Av3rzX9rD4P8AjP4leL/DupeGNI/tCGzspI5JPtUMJWQTI6ffceleDfGDw78WEtdOX4h3ExjLTNYtdz2svlt5fzv+4HT7lfz9i8TXy+tiq0Yfy+9ynxGKxNfLsViq0aPx8vLPl901Pg38VNM8HfHTxT4xu4L1tN1EaiY/JgTevn3qTJv+f+4le8fHnVPD/jD9l/WPGOlaVFBLqkNrcJLPaJHc5e5jXL/7f1NeIfFnXvhVefC3RbPwXFZx+KEmtnvmg0uWB3SOF9/zvHs+/Vfw/wDDn43eNvhrpun6alxf+DL21Q21vJfWscbwbw6ZVxv6YrzKOKrwhWwb/fc8fs/3jgpYqtCnWwSXtuaN/c974j3v9mltv7LN9/111P8A9GyV8p/tcf8AI1eE/wDsFz/+j6+xPgx4L1vwF+zrqGj6/Z/YdRjOoXDQeeknyPJI6fOnsa+O/wBrj/kavCf/AGC5/wD0fX1PC1OdPi/LKdT/AJ9S/wDSZH22Hpzp5fRpz/lieHV7zoen2tvpWmulrDHP9lj/AHkcCRyf6uvBq+gNH/5Aumf9esH/AKLr+vsP8RzTLdFFFeqZhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFWoIEjT7Tcfc/gT+/Ut8oBb26Rx/abj7n8Ef9+obi4e4fe9FxcPcPveo6hJ7sAooorUAooooAvR/wDEzj2P/wAfX8En9+qMkflybHoq9HImpx7H/d3X8En9+sP4foMo0USRvHJsf79FbiCiiigAooooAKK6Sz0O2s/MS4tku7tI/MnkkuvIgh/6Z7/79UNY0dLeHzoUe32fO8G/fHs/vo9cEMZCpPkRp7MyqPMqS3t3vJ0hiR5Hf+COutt7f/hEvDv22ay8y+d9j7/4KuviFS0W46cDnPD94mn63a3T/wCrR/nrpL9NQ0261nZZTahBqafI9v8AwVU+z2fjK1ke2RLTVYfvx/36ybTX9W0eOS2SZ49nyeXIm/ZXm1IPETuvjRp/DJNU8OPomn2s1zMn2ub/AJYVkjipbi8nu5pJ7mZ7id/+WklRDmvXoKoofvNznFopkknlxyO/+rSse18NeMfizBff8K4gn1CfT4P9Kxcw2uyR9/kf6/Z/ceufGYung6ftKjLp0/aE3hvwvqXx41RvD3hZ4bPUrOF9RmfVppIE2JJs+/H5nz73Sut/4Yp+Kf8A0FvDP/g3n/8AjFfVnw/+DvhH4eXEeo6L4Y0/R9Zmsktbq7s0+d/uO8f/AH2ldvX5Dis9xNeq503ZHswwsPtnwTrn7Mnj74Z6Tc+Kdbv9EuNK0dPt11BYanPPO8cf9xPITfWZo+uQa5BJPCk0ex9n7yvvjXND07xVod9pGsWMOoaTqEHkXVpcf6uaP/nnXw3+094fsPhf8WNO0Twnax+HNJudItbua0sP3cbzvPPHJJ/3wiV7uSZ3U9p7Cvrc5q9D2fwEDULWdrHiCx8PyR/bHePfv2eWjyVpRyeYm9K/S/aQ+A8wKKKK1EFFFFABRRRQAVPb273D/wDPNE+/JJ/BRb273D/880T78kn8FSXFwnl+TD+7g/8AQ6xbeyASedPL8mL5IP8A0OqtFFapcoBRRRTAKKKKACiiigAooq1BAkafabj7n8Cf36lvlALe3SOP7Tcfc/gj/v1DcXD3D73ouLh7h971HUJPdgFFFFagFFFFABRRRQAUUUUAXo5Ev49kz+XdfwSf3/8AfqjJG8cmx08t0oq39oS7h2Sv5c6J8kn9/wD2Kw/h+gypRRRW4gojk8uRH/uPvoopAdddwf2nBPNbwTXdjczJdpJBseSF/wC46PWbr8n2O1S1/wBRO8zzvBH/AMsUf/lnWPp8d1JdJDZ7/Pf/AJ5v5daNheTeFtXnS5h8z+CZP76V4XsXSnprY6ecv+AJUW6v0TZ9reD9z5n+f9yoNP8AE97p93PbavvuIH+SdJP4KTWNJ/s/y9W0iT/Qfv8AmR/8saNX1ay1rS455k+z6qnyfIn36fLCpU591P8AACDVrBNBvYLnTrn9xN88Lxv9ys27upr+6e5uX8yd6Z5dFetSpezWurOcKKKK3EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFTwwPcP/zzRPvySfwVBU73XmQpCqeWn8f+29TK/QB886eX5MXyQf8AodVaKKaXKAUUUUwCiiigAooooAKKKKACk3UtW9L0e91iTZZweZ/fkk/1aVjUqQp6zGU+aWu30/4dwx/PeXT3En/POP5I62LfwjpMH3LCGT/f+evOnmNFbK50ewPMKTbXrEnhnTJP+Yfbf9+ErNvPAemXH+pR7ST/AKZvWcMzg90HsDzndS1s6x4QvdHjkmT/AEyD/npH/rErGr1KdWFXWDOcKKKK3EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVbt7dI0+1XP3P4I/wC/Rb26Rx/abn7n8Ef9+oLi4e4fe9Y/xNEAXFw9w+96joorVaaIAooopgFFFFABRRRQAVdjkS+j2Sv5d1/BJ/f/AN+qVFTKPMASRvHJsdPLdKKtfaEu4Qkr+XOifJJ/f/2Kq0Rbe4BRRRVAEcnlyRv/AHH31115D/acE81vDNd2NzMl0kkGx5IZP7jo9cjT9PjupLpIbTf57/8APN9lefiKPtPfua05m1r0n2O0jtf9RO87zvBH/wAsY3/gqfwBIi3V8ibPtTwfufM/z/uVQsbubwrrE6XMPmfwTJ/fSrGsaP8A2f5er6RJ/oP3/Mj/AOWNcW1P2H83U0/vhp/ie90+6ntdX33ED/JPHJ/BVfV7BNBvobnTrn9xN88Lxv8AcqfV9Ws9a0uOeZPs+qJ8n7tPv1h+XW+HoX996eRGo+6upr66e5uH3zvTKKK9VJJWRiFFFFMAooooAKKKKACiiigAooooAKKKKACrWk6lc6Lq1jqdps+1WcyTw+Ym9N6VVFNY1nOCqRcJbD1Pq74G/HP+3NG1WTxpr+j2dzHe7LZWdLT9z5cfVHf++9d58UPg14c+LUFhH4hhnnjsXeSFYLmSHlxg52EZ6V8PPeho4FRPLjR0fZ/fkr6z+A/xs1L4kXWvRa3a2GmpYJA8Jgd/n3+Zvzv9NlfivEvDUYQniacb038Ue21j2aM4Yin7CsUF/Yz+GLdLDUv/AAa3P/xyvXvB/hCw8E+GdO0LTEZNOsYRBAsjF3CDp81bUcm5d2QQfapC2Oc1+c0cHQw/8GHIdNDB4bC+9RhymD47/wCRN1vH/PnN/wCgGvzc/a4/5Grwn/2C5/8A0fX6SePP+RN1v/rzm/8AQDX5t/tcf8jV4T/7Bc//AKPrkyf/AJLjBf4Jf+kyFi/gPDq+gNH/AOQLpn/XrB/6Lr5/r6A0f/kC6Z/16wf+i6/qrD/GeDMt0UUV6pmFFFFABSZq5pej3urybLODzP78kn+rSus0/wCHcMfz3l09xJ/zzj+SOuGti6NDSZrTh7Q4fmjmvUbfwnpFv9zT4ZP+unz1Yk8NaXJ/zD7b/vylcX9pw/lNPYnk9JmvRrzwHplx/qUezk/6ZvXJ6x4PvdIjkmT/AEyD/npH/rErpo46jU8jP2czGooor0jIKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooq1BAkafabj7n8Cf36lvlALe3SOP7Tcfc/gj/v1DcXD3D73ouLh7h971HUJPdgFFFFagFFFFABRRRQAUUUUAXo5Ev49kz+XdfwSf3/8AfqjJG8cmx08t0oq39oS7h2Sv5c6J8kn9/wD2Kw/h+gypRRRW4gojk8uRH/uPvoopAdddwf2nBPNbwTXdjczJdpJBseSF/wC46PWbr8n2O1S1/wBRO8zzvBH/AMsUf/lnWPp8d1JdJDZ7/Pf/AJ5v5daNheTeFtXnS5h8z+CZP76V4XsXSnprY6ecv+AJUW6v0TZ9reD9z5n+f9yoNP8AE97p93PbavvuIH+SdJP4KTWNJ/s/y9W0iT/Qfv8AmR/8saNX1ay1rS455k+z6qnyfIn36fLCpU591P8AACDVrBNBvYLnTrn9xN88Lxv9ys27upr+6e5uX8yd6Z5dFetSpez31ZzhRRRJJ5ab3rcRDef8eM/+5JXpX7Af/NRPppf/ALd14nrniC9t9YtNOtrX7RaXTwQPcRwSSbN8nlyfc+SvtT4NfAXRPgTNrq6dreoaodWaBJBq3kJs8nzP9X5caf8APevznijFwcFh1v8A5NHfh4qMvavY9Opa8Rk+NPiLc+dKsz8//PCb/wCLrc8D/E7V/EviSDT7ywt4IJEkd5Egmjk+RP8Abr4mpluJpQ9pNaHm4fijLsVXhQhL35nfyeJ9It9WGlvqVnHqTuifZZJ/n+f7nyV538XP2f8Awr8UL6fxFqdtqc+u2Wl+Ra/ZL2RE+Te8fyJ9/wCd60/HPg+PTdQvfGcFxcS39miXS2u1PJaRI9npvrX+HPie98XaTd3F/bQwPDdeQiQI6fJsT+//AL9YKm6cFiaEv+HO2jj5/XJ4PEw9/wCx/hPlz9n/AOAl18XBq5+Knh3XtHezjsvsAkt59LLyTCTz+/z/AHI/9yvIfBniTUdV1S7tbl0kgh8zZHGnl/ck8uv0ytR/pUX++lfl98P43t9c1J5oZreP9/8AvLiB44/9f/t19lkOPr4rFVJVZ/ynXXowpwPQqKI/3n3KZ9oTzNm9PM/5576/ULnlD6KKKYBU8MD3D/8APNE+/JJ/BUFTvdeZCkKp5afx/wC29TK/QB886eX5MXyQf+h1VooppcoBRRRTAKKKKACiiigAooq1BAkafabj7n8Cf36lvlALe3SOP7Tcfc/gj/v1DcXD3D73ouLh7h971HUJPdgFFFFagFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFSWl4lpeQTTQpOiP/q5P46jrp9H8nT/AAbfXqQ/aJ3+R/8AYrkxFT2cPU1pmt5llpmj3eraLa+ZJN/45XL+H7OHX766gu5pvtUyb0eoNF1ubRJt8P7yN/vwf363LXxBoelySXtjp8325/8AlnXl+zqYfnUNb9TT+IY2mavc+HZp4diSJ5jpNBJ/q6zWpZJHuHd3/wBY773pBXr06ah7/U5xaKKK6BBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRUlraPf3UFtD9+Z9lJtJXYGz4X8Lvrc3nTfu7FP/AB+vRre3SzgjhhRI40/5Zx1HZWaafZx20Kfu0TZXNeNPEj2f/Evs38ud/vyf3K+SnOeOrciOz4IGlrHjCy0l/J/4+J/+ecdcvceP9Rk/1NtbW8f/AE0+esLS9Ln1S68m0TzH/wDQK7PT/h3bRx/6ZM88n/POP5ErudHC4XSpqyPfmYsfxA1RPvpbP/wCtrT/AIiWskmy8he3/wCmkf7yOr8ngfRfL/49vL/7bPWDrHw/kt45H06b7R/0wk/1lZ82Cr6W5DT3zuLe4juII3hdJI3/AOWkdcX4w8LpbrJqFnH+7/5bxx/+jK5/Q9cutAuvk3+Xv/fwSV6hZXkOqWMc8P7yCZKxnCpgZqfQX8Q8goq/4g0v+w9UntU/1H+sT/crPWvpKdT2kOc5BaKKK2EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFW7e3SNPtVz9z+CP+/Rb26Rx/abn7n8Ef9+oLi4e4fe9Y/wATRAFxcPcPveo6KK1WmiAKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAElpeJaXsE00KToj/AHJP467kSWWmaPd6totr5kk3/jlZOj+Tp/gy+vUg+0Tv8j/7FY2h65PolxvhHmI/34P79eNWh9Yc+T7H4nRD3CfQLOHX727hu5pvtcyb0kpml6zdeG554NiSR73jnt5P9XWzb+INC0tpL2x0+b7c/wDBXJvK91JI7/6x38x60pRdZy54+4R8AtFFFeoZBRRRTAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACoLyzg1CCRJkSTenl/vE8yp6KTSasxn1l8C/jcvjbUbfwqmhtYRWGmh/tRud+/Z5aYCbB/fr3X0GOK/N2zvLnTX3W11dWkn3PMtJ3gk2f8Ar6f+A/xt06+svC3gieHVJ9bNr5D382x4XkSN3f5/M3n7h7elfjPEHDrw18Vhl7n32/vHuYbE39yZ7N47/5E3W/+vOb/wBANfm3+1x/yNXhP/sFz/8Ao+v0i8d/8iZrf/XnN/6Aa/N39rj/AJGrwn/2C5//AEfX5Pk//JcYL/BL/wBJkaYr4Dw6voDR/wDkC6Z/16wf+i6+f6+gNH/5Aumf9esH/ouv6qw/xnhTLdFFFeqZiHmt3wv4Xk1ubzpf3din/j9Y9nZvf3UFrD9+Z9let2Vmmn2kdtCnyImyvIx2J9jH2cNzppwJLe3js4UhhRI0T/lnHWRq/jCy0l/J/wCPif8A55x1m+NPEj2n/Evs38ud/vyf3K4zTNLm1S68m0TzH/8AQK8/DYNVIe1rvQ0nP+Q3bj4gajJ/qba2t4/+mnz0yP4gapH99LaT/gFbWn/Du1jj/wBMme4k/wCecf7uOr0ngfRfL/49vL/7bPWntsGtOQOSoUNP+IltI+y7ge3/AOmkfzx11FvdR3MKPE6SI/8Ay0jridY8APbRyPp0/wBo/wCmEn+srC0PXLrw/dfJv8vzP38ElZ/VqNeHPh2HPyfGdB4w8LpbrJqFmn7v/lvHH/6Mrj/u16/Z3kOq2Mc0P7yCZK8v8QaX/YmqT2qf6j/WJ/uV2ZfXf8GZnOBRooor2zmCiiigAooooAKKKKACiiigAooooAKKKKACiirUECRp9puPufwJ/fqW+UAt7dI4/tNx9z+CP+/UNxcPcPvei4uHuH3vUdQk92AUUUVqAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVJaXiWl5BNNCk6I/wDq5P46jrp9H8nT/Bt9epD9onf5H/2K5MRU9nD1NaZreZZaZo93q2i2vmSTf+OVy/h+zh1++uoLuab7VMm9HqDRdbm0SbfD+8jf78H9+ty18QaHpckl7Y6fN9uf/lnXl+zqYfnUNb9TT+IY2mavc+HZp4diSJ5jpNBJ/q6zWpZJHuHd3/1jvvekFevTpqHv9TnD7tY3ijxInhO0t55oEkSd9n7x/LpfEniy18KRxvcwTXEeyST9x/sV7v8As9/s1694J8UX2s+LJ9D1zSL7TPLtrWGWS6kjkeRH8zZNDs+5XhZvm8MvhZfGdFGj7Q+Y/wDhcGnf88If/A5KveHfEtr8TvEOleDoXhsJPEF0mnR3ck6T+Tv/AI9n7vf/ALlfol/whvh3/oXtG/8ABfB/8RTrfwlodncrcW2haXb3ELb0njsIY3R/9/ZXw9TibEVKbglY7PqRxfwA+D9z8DvCGpaNLrkesSXmqPqXn2to9rs3wRps2b5P+edanxI+Gv8Awn/2EC9SzNvHMP3lr5+/fs/2/wDYrt6QnFfJxr1YVfbxfvm2KwmHxmHlhqsOaIsZaONE3N8iU7c/9+m0VkdUYRjayCnU2imV1uFfNf7fVwsPwt8KtcS4Q+IwMzv/ANONxX0meoqrqek2WrW6R39lbX8aPvSO6hSfY/8AwOtKFT2FaFTsTOHPDkPzms9cj8P+AdM1F0SSDyII/wDWbI/n/wBuuj0/4Tv4n+Fd38a4tatorTT0nnGjfYfMeT7K5g/1/mfx4/5517Z4P/Zn1zRP2gNR8XX8+g3nhCa5vbiHTPMd5FjmT9x+4eDZ8n+/Xon7RGnWum/s5/EC0srWG0gTSJ9kFrCiR/f/ALiV9nis/nXnRo0NPhPPhhfjnM+PvDmt/wDCQWs8/kfZ/Lfy/wDWb6165D4fyfZ9Dv3f+CeST/yHWz4f8SQeIPtHkwTW/k+X/wAfFfpFGtzwhznlmtRRRXcIKKKKACiiigAooooAKKKKALVvAkCfabj/AFf8Ef8AfqG4uHuH3vRcXD3D73qOsknuwCiiitQCiiigAooooAKKKKACiiigAooooAKKKKAEapbe2mu50ghR5JH/AOWcdFvG9xMkMKeY7vsSu58P6HBoM06PewyalMnyR/3K4MViVQjpuaQh7Q5DUND1DTY99zavGn/PT79WPDmsTaXfRwon2iC5fy3gq1aa/qGk6hPbavvvIH+SZJP/AGSo7+3TwzqFrqGnOk9rNveHzK5PaTqQ9nUW+xqVfE2lpo+tzww/6iSPzEj/ALlZnepLy9mv7p7q5fzJ3qNa9OhCcIWmc4tFFFbiCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArpPh/b/aNcef8A54w1zddf8OP9fqP/AAD/ANqVwY12oSZrT/iHaXEiW8Ek7/cRN9eQXFxJqFw8z/vJ5n316n4g/wCRf1LZ/wA8X/8AQK8z0PZ/bGm7/wDV+eleXl/uQnUOisekeH9HTQ9PjhT7/wB95P771neIPGkOlzyWtsn2udPv/wDPNK3dRuHs9Nupk/1iI714+n99/wDWVlhKP1qbnUCc/ZnUx/ES9jf99bQyJ/0z+Sur0PxBa63D+6/dyJ9+CT/WJXA2nhTU76x+1Qxp5b/cjkf53qjpeoPpGowXSfu9j/PH/sV11MLQqRfsN0Z88/tnX+NvD6Pb/wBpwp86f67/AG46q/D/AFTy7ufT3/1c3zp/v128saXFvIj/ALxHSvKtD36f4itE/wCWiXXkf+yVy0Ze2oTpz6Fy9yZ1nxEs/MsrS6/5aI+z/vuuEHU16T45/wCRbuv99P8A0ZXm9ehl0v3LRlW+MKKKK9cwCiiigAooooAKKKKACiiigAooooAKt28CW6farn/V/wAEf9+iC3S3j864+5/BH/fqC4uHuH3vWP8AE0QBcXD3D73qOiitVpogCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRUlvG9xMkMSeY7v5aUm7asAt7ee7mjghR5JH/5Zx1a1DQtQ02PzLm1eOP/AJ6f6yuv0DQ4NBnnR72GTUZk+SP+5WFZ+INR0jUJ7XV995A/yTJJ/wCyV47xc6k7UVodPIVfDmsTaXfRwIn2iC5k2PBUXibS00nW54Yf9Q6eYkf9yreoW6eGdQtNQ050uLSfe8PmVj3d5Nf3T3Ny/mTvWtFOpV9tDYz/ALhHRRRXqGQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAJtq3pOrX+hahBf6ZeNY3sPmbJofvp+72VUBoas6lONWPJNXQ+p9QfDX4mHWPg1cWuu6pc3+uXH22ESzwff8AnkCfOibOlfI37XH/ACNXhP8A7Bc//o+vb/hl/wAibH/19T/+jK8f/ap0TUNY8TeFntLKa7jTS598kCb9n7+v4yyXMJY3xKnTqQsqMqkIcv8AhkexUd6J8/19AaP/AMgXTP8Ar1g/9F18/wD8FfQGj/8AIF0z/r1g/wDRdf15h/jPKmW6KKK9UzOk+H9v5+uPP/zxhrv7iRLeCSZ/uIm+uL+HP+v1H/tn/wC1K6jxD/yAdS2f88H/APQK+Txnv4k7IfAeWXFw+oTvO/7yeZ/Mr1Dw/o6aHpscKff/ANY8n996820TZ/bGm7/ueeleqahcPZ6bdzJ/rEjeSurHz+CmjOiYOv8AjWHS5pLW2T7XOn3/APnmlY0fxEvY5P31tDIn/TP5K5eP++/+srWs/Cep6hY/aoUTY/3I5H/ePXV9Vw1CC9oP2k6h32h+IbXXIf3X7uRPvwSf6xKwfHHh9JIf7ThT50/13+2lchpeoPpGowXSfu9j/PH/ALFeuSRpcW7o/wC8jdK8+rT+pVlKGxcP3kDh/AOp+Xdz6e33JvnT/frQ+Iln5ljaXX8aP5f/AH3XJaJv0/xFaJ/y0S68j/2Su68cf8izdf78f/oytK1qeKhOHUzj8B5vRRRX0hzhRRRQAUUUUAFFFFABRRRQAUUUUAFFFWoYEt0864+5/BH/AH6TdgC3gSBPtNx/q/4I/wC/UNxcPcPvei4uHuH3vUdZpPdgFFFFagFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAI1S29tNdzpBCjySP8A8s46LeN7iZIYU8x3fYldz4f0ODQZp0e9hk1KZPkj/uVwYrEqhHTc0hD2hyGoaHqGmx77m1eNP+en36seHNYm0u+jhRPtEFy/lvBVq01/UNJ1Ce21ffeQP8kySf8AslR39unhnULXUNOdJ7Wbe8PmVye0nUh7Oot9jUq+JtLTR9bnhh/1EkfmJH/crM71JeXs1/dPdXL+ZO9RrXp0IThC0znFrJ1TXLWzuo9L3vHf3XlxwRxp/wAtHk8uP5/9+tUmuN8QafdSeONFvkhf7Jaz2Mk9xs/dpGk/mSf+OVhipzp0XKBdM+iPgL+zXqU7eIk+MPg6z1NfLtU0tNWngvtn+v8AP8vY/wAn/LCvqSONLeONETy40Ty0jj/gSuf8K/E/wp8QrjUT4W8R6ZrxsmjNyLCcSeTv37N/+/sf/viuhr8FxFetiKjqV3qfQwhCnD3BaKKKyLCiiigAooooAKKKKACiiigAqjrmh6d4k0e/0jWLKHVNNvU8i6tJ08yOaP8AuPV6ikB8jfGj9m/xN/wsLTF+G3ha3svCP2W1+2w2N7BaQST+fJ5/mI773/cbKw/2xPBeh/Cu+8IR+C9MtvDCanBqP2r+yYxB5+x4Nm//AHPMk/77r7PnuYLO3nnmdLeCGN5Hkk/gjSviz9r3xxofxYufBU/grVrPxXHp9rffaZNJk8/yd/keXv8A+unlv/37r38uxWKr4mlTu2oev4nJWhCEJnM2/wDqY/8ArnHTz2plv/qY/wDrnHTz2r9tjseEhaKKKoAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKPMorqrCSHwz4bg1BIUuL65fy0kk/grjxFf2K0RpTgc5pd9Hp+oQXP8ArPJffWx4q015Zv7atH+0Ws3lvvj/AIHq5b6nZeL/APRb6FLO+/5Yzx1m2uoXvhG6nspkSRH+/A/3H/6aVwc06k7pe+vyNR9/r8Ot6Rsvkf8AtGH7k8f8dYdJtpa9SnShT+A5woojkeN43R/LdKuyRpfo80SeXOn30/8AZ0rSUuXcRSoooqwCiiigAooooAKKKKACiiigAooooAKKnhge4f8A55on35JP4KZPs8z91v2f9NKnm1sBHRRRVAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFdJ8P7jy9Ynh/57Q1zS9KsafePYX0F0n34X31yV4e0pzgaUz1u5t0uLd4X+46bK8hkjn0+6dP+W9q9ev29wl5BHNC/mI6b0rk/Gnht5/8AiYWaeY//AC2jj/jr5/BVlTnyT6nTOB0mn3kGu6XHMn3Jk+eOvM9U8P3ul3UkDwTSR/wSRp9+o9I1y50eTfaP+7f78cn+rer3iT44WvhOxgnv9P8A9c/l/wCv8uP7lbuFbAtzh8Bn7kzutEkkt9DtPtn7iRE+fzK8A8WeONRj8Taslhpf2y0kn/cXEdrPJv8A+2ifJXVeF/Gcn7Snii38CWV3beHUvYZrp7qJ/t0nlwoH2eX+7+/X2b8EvAE3wv8Ah7o/hVdSbUIdM8//AEyOD7L52+d5P9Xvf+/XyWOzj6i7UvjOyFH2x852/ijUY4I4/wCzvuf9Os9eeeDNYn8SeN7v7TB9jjS9kkg8xHTfH+8/v1+hl5dNbwP8z+Z/B89fMf7TWnCbxj4Y1CV97waRdQJ5nz/8t4/M/wDQI68nAZ1VdSzW5c6BynxAuPL0WOH/AJ7TJXn1bvivVP7U1VIYf3kFqmxP+un8dYVfqmBhyUTzKnxhRRUd5cfY7G7uv+eMEkn/AHxXpN23MgaRIx87pH/10euF/wCFga3H9/RH/wDAG6rtfhj8IU/aq0/X2g8QWvhz+yY0tJPLtf7R87z0d/76bPuV+gkUksUKJ5rfIiJ9+vzzNuInQr+zw+tj0KOF9ofmN/wsDV/+gI//AIA3VdX4f1iTVNNgnuUS3u33/wCj/wCr2f8AAH+ev0S+0S4/103/AH3XyT+0p8CX0vWvGHxhGvrciIwXX9jCx+cbI4LXZ5/mf8D+5XLl/E0p11TxK0f9djWeF9nA813UtZPhvWH8QWMl08H2fZP5fl+Z5la3l+X9+v0iFSFSPPA8sKKKK2EFW7e3S3Tzrj7n8Ef9+iC3S3j864+5/BH/AH6guLh55N7/AH6x/iaIAuLh7iTe/wB+o6KK1SsAUUUUwCiiigAooooAKKKKACiiigAooooAKKKKAEWUsKsaXfpp+owXP+s8l99dJp8sHhfw1BqCQR3F/cv5aSSfwUW+p2Xi8/Zr+FLO+/5Yzx15UsS583uXga8hU8VaW8k39tWj/aLWby5PMj/geo7/AF6HWtH2XyP/AGjD9yeP+Oo7TULzwldT2UqJIj/fgf7j/wDTSsZqWHoc69/psHOLRRRHI8bxuj+W6V6voZBRV2SNL9HmiTy50++n/s6VSpRlzAFFFFUAUUUUAFFFFABRRRQAUUUUAFFFTwwPcP8A880T78kn8FJu24EFFST7PM/db9n/AE0qOgAooopgFFFFABRRRQAUUUUAeq/DL/kTY/8Ar6n/APRlc78Yv+Qlpn/XjP8A+jK6L4Zf8ibH/wBfU/8A6MrnfjF/yEtN/wCvGf8A9GV/DvCv/J0K3/X2v/7cepP+AfHtv/x6p/uV9CaP/wAgXTP+vWD/ANF1892//Hqn+5X0Jo//ACBdM/69YP8A0XX9nYf4zz5luiiivUMzpPANx5esTw/89oK7+4t0uLd4X/1bpsryPT7x7C+guk+/C++vXLW4S8gjmhfzEdN6V8xmEHTqc52UTyCSObT7qRP+W9q9er6deQ65paTJ9yZPnSub8aeG3uP+JhZp5kn/AC2jj/jrk9H1y60eTfZv+7f78cn+reuicPr1FTh8aI/hkmqeH73S7qSB4JpI/wCCSNPv16P4fM8Wi2v2tPLdE+euct/iRH5f77T33/8ATN6ydc8YXWsQPAifY4H+/wDP+8epqU8TiLQnHYPcpmNqdwl5fXzw/cmnk2V7FHH5duif3K858GaA+oX0d46f6JC/mJ/tvXaeIdUTR9Lnm/j+4n+/WOMftJwoQ6Fw/nOE0uP+0PGfyf6v7bJJXU/EC48vQ44P45p6zPh3pefP1F/+uCf+z1meMtU/tPWNif6i1/d/8D/jrdQ9pilBfYI+wYdFFFfQHMFFFFABRRRQAUUUUAFFFFABRRVqGBLdPOuPufwR/wB+k3YBYLdLdPOuPufwR/36guLh7iTe/wB+i4uHuJN7/fqOs0nuwCiiitQCiiigAooooAKKKKACiiigAooooAKKKKACjzKK6qwkh8M+G4NQSFLi+uX8tJJP4K48RX9itEaU4HOaXfR6fqEFz/rPJffWx4q015Zv7atH+0Ws3lvvj/gerlvqdl4v/wBFvoUs77/ljPHWba6he+EbqeymRJEf78D/AHH/AOmlcHNOpO6Xvr8jUff6/DrekbL5H/tGH7k8f8dYdJtpa9SnShT+A5woojkeN43R/LdKuyRpfo80SeXOn30/9nStJS5dxFKo7iP7RbyQf302VJRVbgZXhf4oa7+zfa6pL4Ys7PWpdVgR5k1WB5Nnkb9gTyXT/nu9ff2ja9a61aQPDe2c872qTvBBMj7N6V8LRyeU9Zfwt8TJ+zb4mvfEsGkQ6/JqVtJpz28b/YdnzpP5m/Y+/wC5X53nuRc18Rh1qejhq3J+7mfZemfELUb74gXWjXNrbW+mwzTx+f5ciP8AJ9z5/uV6BHIkyAxskkf/AD0jeuE0+4b44fB7w/qLD+w21y1tdR8j/j6jh/5aeX/BvrmofiAvwo8zwp9ih1M2D/8AH153k79/z/c2P/fr4n6ssZpQj78fs/8At1/0PG+v1MkqT+vz/cz+3/e/l5Y/3ftHr0lzBbvseaGN/wDnnI6JUv3q+ZvGXjO18Y6/a6m9nDBJCkaeX52/fsd3r2j4e+PH8by6irab9ga3Cfcn3+Zv3/7H+xVYnLK+Foqs16+ROVcUYPMMZPCx0/k/vf8AyJ2VNkljt498zpGn/PSR9lP2t/cavDvid8RDq0Oq+HpdNjght7zYLp5vM37H67NlcmEwtTGVPZ0z1s4zehlGG9tW/wC3Drbr4i6lb/EODQ7e1trjTXnhj+1eW7yfOnz/AD/cr0IDFeKfC34jjTYdI8NRaeHjmuXT7Ylz03vv+5sr2xY3HO1sVpmFB4eapuHLb/ybzODh7HLMKM6zr83/ALb/AHf7wnXmjrUixnlmVgB3rgPA/wATl8Y+ILvT1sEtPJhkn8z7Tv37H2fc2VyRpzqwnOC0ie5UxlDDVoUas/fn8J3lFFFSd581ftKfHzxP8O/GsPhTSdF0/UdM1DRo7ia6uLW5kePzpJoX+eN9n3I6+e/DHhu18NxzwWd7JeR7I4/3jo/3P9yv0O1azOpaPfWCzPbm5heDzP7m9NnmV8DfEv4Sv+y4uh2n9rx+KP7ZtZ/3kdl9h8j7L5af89JN/mef/wCOV9lw7j8Nh5+zqQ999f62PMxUJ/GWaKZHJ5kcb/30p9frR5QUUUUwCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACpLe3e8mSGFPMnf7kdFvA93MkMSPI7/8s462PCMiWniSNLj92+x0/efwPXJXqezg7bmkBdU8IXWj6XJeTXUMmz78cdJpGr2Vxpf9k6pvSDfvhnj/AIKsXl5PaWOq6Zcw3Ul3dXXmQSRx/u3/AM7Kr3/h2HSdF330z/2jN9yCP+CvMhU9rD2dfvoahqFhoen6fPsvHv7p/wDUbJPuVj3FxNeSb5pnuH/56SVFmlr06WH9n1uZe0CiiiuszCiOR45N6P5bpRRQBekjS/R5ok8udPvp/wC1EqjRHI8b70fy3Sr0kaX8ck0SeXOn34//AGolYfwhlGiiitxBRRRQAUUUUAFFFFABU9vbvcP/AM80T78kn8FFvbvcP/zzRPvySfwVJcXCeX5MP7uD/wBDrFt7IBJ508vyYvkg/wDQ6q0UVqlygFFFFMAooooAKKKKACiiigAooooAKKKKACiiigAooooA6Hwr4n/seT7Ncf8AHi//AC0/uV6LFLHcRxujpJG//LRK8Z6Vb0vWL3SH/wBDn8uP/nnJ/q68XFYHnftKZ0QqHoWoeD9P1Sfe6Pbzv/y0gpP2TfhvBpfjzxI890mqRyaWnlx3cCSbP3/+s+euQ1D436d4b8v+2LZ4/k8zzLf95+7SvcvgH4R8UeFPEWqX2uaFJpVnc2CRwu15bTeY4kL/AMEj/wAFfC53VqU6HsJz1O2goVJns1n4X06xbfHY2aP/AM9ILVE/9AStOSRLOP5/3cdc/wCIPF0PhvTjeXx8m23ogdE3v8/+xWHo/wATvD2tanDZwXVzPdXL7E8y2dK+Djh61SPPymlbH4PD1vYTrR5zqZJXnk3v/wAAjryD4+fCnxT8TI9KHhO80i0mhtp7WZ9VnmjKO7j7mxHrd8fWvihPE8Go6fdXFtoVvDA91suvL+5I7z/J/uV1vhXxrpfisXUulTyubdk8wSQun3/uV1qM6MY1qb/4Hqc9HHwr4meGn7k//Sv8J8D+DfB2p/Hy+bRPDM1tpl3psP2qaTVpngR03+T8myN/46bZXn9k65P4VuUeTUtJeexnuI/9Q7wSbJPL/j2fJX3T4P8AhZ4N+H9/NfeGfCuk6Bd3KeRNPYQeXI6b9/l/9914f+1p8O/DHhL4c6h4q0HQNP0fxHdatbefqlpAI7iTz5G88+Z/t/1r67CcQVZ43XaZpUw3uHj+0VxuoSf8XX8Lp/yzk1HTd8f/ACzf/S460vD+sfZ/CMeoX880mzzJJJP9ZJ/rK1fBfwt8VfFfxXo3ivwrpaahoWnapYw3U890kEiPBOk8n7t/9h0r7TNMVTjheec7Hn04e+foHFp9rp7XKWdpbWaO3z+RCib/APvin1LcsGuJNvG564U/GTwt2ubn/wAApK/FaNOrW+CNz2cTjMNgf96nGB2jHbXEfGzwVqHxK+FPiHwzpMtrDf6lCiQSX7ukEf75H+fYj/3KueMG1PxD4Pgm8MzzR3FxJBMkkb+RJ5P/AAOs7wZ4uXT2s/DWt3VxJ4l3vHNuj8z53+dPn+59zZW0aM7e1h8Ufv8AX0PPnmVGGJ+rT+Cf2/s/4f8AEfF3iDwXqnwj1238A6ybS51vV/IeCbTpHkgj+1SeRH5kjoj/AH4/7lSeMvAurfs73EGn+KprbU59UjeeCTSZ3n2RwfI+/fGn/PSvuPxB8LvB3izX7fXta8MaZq+s2vlpBf3cHmTpsk8yP5/9h6l8VfC/wj8Rr2zuPFHhjTfEFxADDA99AJNiO/z19BT4grU/Zye0N/M6/qq6HxPH+8j8yrdvAkCedcfc/gj/AL9cR8M7i6uJLu61W6nuI9n7iCR/v/vP/QK7C4neeTe/36/WaFZ4iFzywnneeTe/36joortWmhkFFFFMAooooAKKKKACiiigAooooAKKKKACiipLeB7yZIYUeR3/AOWcdK6WrALe3e8mSCFPMnf7kdbGqeEbrSNLkvJrmGTZ9+OOl8HyJZ+JY0uP3b7HT95/A9T3l5PZ2Wq6Zcw3Ml1c3XmQSRx/u3/z5deRXxFT2yhDY6IQK+katZXGl/2Tqm+ODfvhnj/go1DT9C0/T59l69/dv/qPLf7lGoeG4tH0XffTP/aM33II/wCCsOrp04VG505snUknnmupN8szzv8A89JKjoor1ErKyMQooopgEcjxyb0fy3Sr0kaXyPNEnlzp99P/AGdKo0RyPHJvR/LdKxcW9QCirskaX6PNEnlzp99P/Z0qlWkZcwBRRRVAFFFFABRRRQAUUVPBC87/APPNE+/JJ/BSbtuAW9u9w/8AzzRPvySfwVJcXCeX5MP7uD/0Oi4uE8vyYf3cH/odVKySb1YwooorYQUUUUAFFFFABRRRQAUUUUAeq/DL/kTY/wDr6n/9GVzvxi/5CWm/9eM//oyui+GX/Imx/wDX1P8A+jK534xf8hLTf+vGf/0ZX8O8K/8AJ0K3/X2v/wC3HqT/AIB8e2//AB6p/uV9CaP/AMgXTP8Ar1g/9F1892//AB6p/uV9CaP/AMgXTP8Ar1g/9F1/Z2H+M8+Zbooor1DMQ10XhPxR/Y7/AGW5/wCPF/8Alp/crnSM0YxWFajCvDkmOn+7PZ4pY7mNHR0kjf8A5aJWNqng/T9UkkfY9vO//LSCvPdL1i90h/8AQ5/LT/nnJ/q66iz+I5/5ebL/ALaQPXz8sHXoSvTOznhMP+Fbn+DUH/781e0/4f6fbyb7mR7t/wDpp9yo/wDhYtn/AM+t1/3xH/8AF1nXnxEnkj/0OySP/ppO++j/AG2ppqHuHYXmoWuj2O+Z0t4ErzXxBrk2v3W9/wB3An+ojqpeXt1qE/nXk73En/TSq/WvQwuD9j789zmnM6mTxRDp+gwafpv+v2fPJ/c/v1y5ooruo0IUfgDnCiiiukzCiiigAooooAKKKKACiirUMCW6edcfc/gj/v0m7ALBbpbp51x9z+CP+/UFxcPcSb3+/RcXD3Em9/v1HWaT3YBRRRWoBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABUlvbveTJDCnmTv9yOi3ge7mSGJHkd/+WcdbHhGRLTxJGlx+7fY6fvP4Hrkr1PZwdtzSAuqeELrR9LkvJrqGTZ9+OOk0jV7K40v+ydU3pBv3wzx/wVYvLye0sdV0y5hupLu6uvMgkjj/AHb/AOdlV7/w7DpOi776Z/7Rm+5BH/BXmQqe1h7Ov30NQ1Cw0PT9Pn2Xj390/wDqNkn3Kx7i4mvJN80z3D/89JKizS16dLD+z63MvaBRRRXWZhRHI8cm9H8t0oooAvSRpfo80SeXOn30/wDaiVRojkeN96P5bpV6SNL+OSaJPLnT78f/ALUSsP4Qyg1YfizQ7rxBa2kdtJDHsn3v5lbtFOdP2kOSYHpXw6/a28L/AAx8B+HfCOo6F4ivL3Q7CDTp57CK18h3jAG9N9xv2V1/hr9r7wT408XaRodt4U1xL7VbxLKOe7tbLy0kf/np+/8AMr5zk8N6XcTPI+nQySO/mPJsqvcaJHptvJe6LBHp+sw/PZXcHySQSf30f+Cvg8RwvD36lN6nX9YT3R+glxpdrdafOiWttHI8Lon7lP7lfPfibwTqngVbGO8uYXkuUf8A48Z3/g/74/v1ifA/9o6w+HnhfUtN+J3iLV7/AFqfUZLi2kkhn1H/AEXZGifPGPk/eJJ8ley+CviL8Pvj498ulq2snRthmGo6e8Pkibfs2eYP+mL18lQqYvK6kvaw9zqeRnWS0M3o/utJ/YPMfCvh3UfF+pT2Vne+W6Q+f+/nf+/Xongv4W6noPiiC/v5bGa1RH3xxyO8j70/20rS8aeKvAvwHtLXXdTtRpMOoTDTkmsbB5pHfZI+z93/ANczXzZ4X+O2s6N8br3xbrvijWpPhnNdXsltZySSTx+RIg+zf6LjzP8A4iu2eKxeYRm8JC0OU8zLeF8NglCeLfPOEj3P4meAdRXUdY1+FrODTYoUfZG7pJ8kez+5XnFv587/AOvmj2ffk89/kr0vQP2lvhr8SNXtPDVnd3+oSam32VIZ9JnSFy/995BXfnwF4Y8vYmh2Hl/9cKxwuaTwlP2OJhscmZ8I/XMT7fAzsp/+lHinh/wvqnjaK9ttLvFgW3CF/tVzIm/fv/uf7lfQVjYw2UCbIYUfYiPJGlV9E8N6V4faRdLsLawe52JJ5CbN9eR/8NmfCve6f2tquUd4/wDkC3f+Fedi8TWzCp+5h7p9PkmTQyej+91nI9sJ5pTXxT4X+O2s6V8bL3xbrvijWpPhnNdXsltZySSTRiCRB9m/0XHmf/EV6H8Rf2sPCHibwDr+j+Ddc1iDxVfWMkGmXEdhPayRz/8ALP8AfyD5KynluKhOEPZ7n0ftoHcfFj9pnw58HfFcGg6vpGuahdTWKXyT6bHBs2PI6eX88kfz/uK+bvj/APE/T/2kLjw/J4cstQ0v+woLqOf+2tke/wA/y/L2eRJJ/wA8Hrlo7bU/ErSXvjKabX9WT9xBd6lN58nkf889/wDc3vJWhp+l2Ol+Z9jtUt/M+/5affr73LeG4UHCvVeqPLqYr2hYt4/LgjT+4lPoor9BOIKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAQtit630S11jQ0m07/j+hT99BI/36wiuVroNQ0//hFE065tpn/tF/neP/lnsrzsQ3eHL8ZrAj8E3cNrr373935ieWnmf36zdYs7qw1CdLv/AF+/zPM/v/7dbOr29t4i0t9WtE+z3cP/AB9QVVk15L/RXttRhe4uk/1M8dctOc3U9pbyZoSW/jbU4INm+G4/6aSJ+8rJu7ya/unmuH8yd6r4pa9Gnh6dN3gjn5wooorqEFFFFABRRRQAURyPHJvR/LdKKKAL0kaX6PNEnlzp99P/AGolUaI5Hjfej+W6VekjS/jkmiTy50+/H/7USsP4QyjRRRW4gooooAKnt7d7h/8Anmiffkk/got7d7h/+eaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MXyQf8AodVaKK1SsAUUUUwCijNGaACiiigAooooAKKKKACiiigAooooAKKKKACk3UNWV4g1SezsZP7NT+0L7fH/AKPbxvPJ5f8Ay0/dx/PWNSpCnHnmMkuPEGl2c0kM17DHOn345HqvJ4ksbyOSGwvYbi/m/d2sEf7yR53/ANXGn+3v8uvWfhX+yzoHxd8B6X4u8Qap4l0bWNSM/n2Fp5ECJsnkgT93PA7/AHEruNN/Yf8ABWj6tYahB4k8UST2V1Ddokk9r5byI6SfP+4/6Z18JW4ooxnOCR2Qwszz/wCBH7PR+I1r4jHxa8JarA0DwwWH2uSfTi6SRv5+zyJE3/Psr7Cj/dxon/LNNiU93DM7/wB+kr87xGIqYqo6lR7nrxp8iseTw2fiLxZ4pudJ8SWV4/hvzpnXEPkxjZJ+5/eJ89dfpfwx8N6TqEF7a2UyXVu+9H+0vJXUcUjZ7UTxM5e7D3fQ8XDZPhqb56y9rP8Aml70iK+tUvLWa2mTzIJkdH/3KzPDPg7SfB8c6aZbPAk2zfvmd/uf79bNLXPeXLy3PWnRoVJwrTh78Arxf9rXw5q/ir4NzafoWkXmuX39qWT/AGSxh3ybEk+d69opMZp05exmprdGs4e0PgL4X/CPxzqnjTw5oPifwXr9v4PnuSl/59q8MaQ+XJJ886Hen7zy6+1fh58NfD/wu0O40nw1ZTWFlPcvdSJNcyTfv3RE+/J/uJXU0V14zHVsY71GZwowpnn/AMTdU8W2dxZDw1BdSR+TI832e2Sb5/8Aln9+rkfwh8LeWm+wm3/x/wCmz/8Axddkwz7GlOccVksTOEIxpe76aX9Ty55TRrVp18T7/P8AYn73L/h/xFbT7GDS7GCytk8uC2RIEj37/kSsuTwLo0niL+3Xtn/tTf5nn+c/3/L2fc/3K3dvvRWEZyjflkepPDUJwhCUPgFqW1+S4hd/79MptRvobn5t6ToPiPwLd3F74y0bUPDkF5+4huNWh8iN5PM3+Wn/AACt6TVLK3sY7p7pI7ST7lx/yzr7E+MHwR0T41abpVnrWo6pp8OmXL3SSaTJCju7ps+ffG9fI/gz4cy+OPi9ffDfUYNZs/DGnz6hBDqUNrJHO8drJ5cH7x4/I+f/AK5/7lfpWVZ/D2E1V3geRUwvvlezvIdQgjntp0uIH+5JHU2aoeNbFPhj8XNR8A6c7XmlafdRwJeX3/H2++0jn+fZsT78n/POrjXEEcmx5o45P+ecj19ngcXDGUFXXU4pw9mSUUUV6ZmFFFFABRRRQAUUUUAFFFFABRRS/wAFAG3b6JbaxoiTad/x/Qp++gkf79Hge8hsde/ffu96bE8z+/Umoaf/AMImum3NvM/9ov8AvHj/AOWeypNYt7XxHpb6vaJ9nu4f+PqCvD53KPL9iZ0mNrFndafqM6Xf+v3+Z5n9/wD6aVo23jjU7eHy98M//TSRPnpkniBNQ0R7XUoXuLtP9TPHWNXZTp+0hyVobGf+AnvLya/unmuX8yd6rnNANLXckqasjMKKKKoQUUUUAFFFFABHI8cm9H8t0q9JGl8jzRJ5c6ffT/2dKo0RyPHJvR/LdKxcW9QCirskaX6PNEnlzp99P/Z0qlWkZcwBRRRVAFFFTwQvO/8AzzRPvySfwUm7bgFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6Li4Ty/Jh/dwf+h1UrJJvVjCiiithBRRRQAUUUUAFFFFABRRRQAUUUUAeq/DL/kTY/8Ar6n/APRlc78Yv+Qlpv8A14z/APoyui+GX/Imx/8AX1P/AOjK534xf8hLTf8Arxn/APRlfw7wr/ydCt/19r/+3HqT/gHx7b/8eqf7lfQmj/8AIF0z/r1g/wDRdfPdv/x6p/uV9CaP/wAgXTP+vWD/ANF1/Z2H+M8+Zbooor1DMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooq1DAlunnXH3P4I/wC/SbsAsFulunnXH3P4I/79QXFw9xJvf79FxcPcSb3+/UdZpPdgFFFFagFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAIWxW9b6Ja6xoaTad/x/Qp++gkf79YRXK10Goaf/wAIomnXNtM/9ov87x/8s9lediG7w5fjNYEfgm7htde/e/u/MTy08z+/WbrFndWGoTpd/wCv3+Z5n9//AG62dXt7bxFpb6taJ9nu4f8Aj6gqrJryX+ivbajC9xdJ/qZ465ac5up7S3kzQkt/G2pwQbN8Nx/00kT95WTd3k1/dPNcP5k71XxS16NPD06bvBHPzhRRRXUIKKKKACiiigAojkeOTej+W6UUUAXpI0v0eaJPLnT76f8AtRKo0RyPG+9H8t0q9JGl/HJNEnlzp9+P/wBqJWH8LToMo0UUVuIyNU8M6drF1595A8kmzZ+7ndKfpPibxV8Jp5v+FfXU+n/2mkf23y7ZLrf5f+r/ANej7P8AWSVp5FLXn4rA0MTDkqQujWFT2Znat448dfFSODS/H17cajpVq/2u1SWzhtSk/wBzf5kKJ/BJJUsfhm01DTY9LdP9Bh8v+P7mytO3ge4f/nmiffkk/gqS4uE8vyYf3cH/AKHWGGwVDCU/Y0oBOp7QwZNPj8KrBqHhmOa31XTH+1WT/wCsk89PuPsf7/8A1zrSH7QvxwH/ADG7r/wS2X/xijcKWufE5PhMQ71KdwhWnTGzftEfHNY5GXW7ren/AFBbL/4xWBH4D0eSPe9rJ5kn7x/37/6yuhorTC5VhcHrTgE6lSoUJNAsrjS49OdH+yJ5eyPf/cqhZ+DNIs7qOeGCTzIX8yP9+9buRRkV6nsaf8pmLRRRW4gooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACrUE6Tp9muPufwSf3Kq0VLXMBJcW7wPsf79R1at7hJ0+zXH3P4JP7lM/s+b7V5Oz56zVTowIf4K6j9z4wsoE85LTVLZNn7z+OuauI0jfYj+ZH/z0qPy6xq0vbWa0ZrTOtkgg8JaJewy3KXF9eJs2JXJUeX5f3KKMPQ9ldt6sKkwooorsMgooooAKKKKACiiigAooooAKI5Hjk3o/lulFFAF6SNL9HmiTy50++n/ALUSqNEcjxvvR/LdKvPGl+kk0KeXOn34/wD2dKw/hDKNT29u9w//ADzRPvySfwUW9u9w/wDzzRPvySfwVJcXCeX5MP7uD/0Om23ohCTzp5fkxfJB/wCh1VoorRLlQDWmiRdzP5aVD9vtR0uof+/ldn8I8yfFLw3H2+1P/wCiJK3f2hP23/h5+zH46tPCninw3rmoX11p0OpJPpNrayQbHeRP45E/551+dZ9xW8kxkMLGjzuceb4rfoexg8v+tUefn5DzKK4S4+68cn/XN6k4ry/4vf8ABST4YeN7rSptI8MeKIEtYJ43jmhtYM+Y6H/lnJ/0zr6kn/Zt8QW00kf9taR8n+xP/wDG66MLxdl08NCtjJ+ynL7JnPL63PyUffPK8D0owPSvUf8AhnPxD/0G9F/74n/+N0f8M5+If+g3ov8A3xP/APG67P8AW3I/+gj/ANKD+zcZ/IeXbh61FJcQwfelSMf9NHr1Y/s5eIW/5jei/wDfE/8A8br0b4T/AA7uvAdjq0GpXWn6hJdTxyJ9nR/k2R+X/HXlZhxtleFw062Gn7af8v8ASOihlWJqT5Jw5D5hXULbvdQ/9/KkjvIHk2JOkkn/AEzevf8A9oj9orwj+zH4U0rX/FOi6hqdjqd4dOSPRoIHkRxH5nz75E9K8A/4bv8Ah3+0Z/xR3hbw/wCI9M1V/wDTjcala2qQBIf3j/ckd68vK+OamZV4UvqvIpf3v/tTTE5T7CE585JRRRX64fPk9pIkfmJKnyP/AMtP+WiUTwPBJ/z0R/uSf36gq3Z3iRx+TN+8gf8A8crF3WqAqUVPcWb28n/PRH+5J/fqS3t08vzpv3cCf+P0/aK1wIo4H8l5vuJ/6HUNT3E7zyf880/gj/uVBH9+OqV7XYGF4w8UJ4PtbSeZI/Lmk2f6RP5f/LOve/2b/wBnc6LrXhf4pR+JUmi1bSnvv7IXTguz7bDHJs8/zPn2f7nz1wH7D9nDqHxK8VR3kCXkaaLG6R3aef8A8vX+3X21HGltHGiIiRp8iRxx7ESvyTPs2rV6jw8NEvxPZwtH7ZI7O/3n8ym0UV8ed4UUUUwCiiigAor5j/aX/b68E/ss/EC08JeJfD/iLU7650uDVUuNJEHl7JJJE2fPJ/0zrrP2Vv2t/Cn7W0fiSbwvousaTFoU1rDO2teT8/n7+I9jv/zw/wDH65vbRHY9wor4O/4fGfCT+Pwb423/AEsv/j9J/wAPjvhF/wBCZ42/792X/wAfo9tAv2Z95UV8H/8AD4z4Q/8AQm+Nv+/dl/8AH6+lf2aP2kvD/wC1N8Pb7xd4a0vUtHsLPVJNKaDVtnmb0ggk3/JJ9z9/HRGtGRFj12m06iukQ2nUU2kAUUUUwCl859u3e+z+5vpKKQHyd+0z8Djoer+MPjC/iJWjjMF7/Yz2OzH7qC12GfzPbf8Ac715j4J+EY+M3gTxD8R4Nbh0yLw+80H9mx232rz/ALLCk5/f+ZHs379n3Pkr79kjjuYXhmRLhH+/HIm9HrlPiPZwWHwp8ZJbWsNnH/YuoP5cEKIn/Hq/9yvXo5tiaVD6tGX/AA3Y5J0Yc/OfCnhPxQ/ii1nn+y/Y9nl/u9/mffrdrifhP/yB7v8A7Yf+i5K7UGv2nCVJ1KKnM8MWiiiu4QUUUUAFFFFAFu3uEnj+y3P+r/gk/uVBcW7wSbH+/UdW7edJ0+y3P+r/AIJP7lY/w9UBUpf4Km+wTfavJ2fPTJ40jfYj70/56VpzJ6AdP+48Z2MCeelnqtsmz95/HUklvB4Q0S+hmukuL68TZsjrkBGaDFs+5Xm/VH8PP7hrzhRRRXqGQUUUUAFFFFABRRRQAUUUUAFFFFABHI8cm9H8t0q9JGl8jzRJ5c6ffT/2dKo0RyPHJvR/LdKxcW9UAUVeeNL9HmhTy50+/H/7OlQW9u9w/wDzzRPvySfwUe07gFvbvcP/AM80T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVShJvVjCiiithBRRRQAUUUUAFFFFABRRRQAUUUUAFT2rpH5iSp8j/8ALT+NKgopNX0A9b+HNv8AZ/CESf6xPtU+ySP+P95XL/GL/kJab/14z/8Aoyuo+FUn/FExo/8Aq/tU/wDwD95XN/GWPy9S0z+59in/AHn/AG0r+IOFY/8AGz63/X2v/wC3Hqy/gnxzb/8AHqn+5X0Xo8D/APCO2E3+rT7LB/wP93HXgej6en2GO6vP3doif9tJ5P8AnnHXvlnePeaTpr/6uP7LBsjj/g/d1/ZeH+M86ZJRRRXqmYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVPPafZ/LdH8yB/uSVBU9peeRvR08yB/vx1k77oCCip7iz+z/Oj+ZA/3JKkgt0t4/OuPufwR/36XtEAQW6W6edcfc/gj/v1BcXD3Em9/v0XFw9xJvf79R00nuwCiiitQCiiigAooooAKKKKACiiigAooooAKtQTpOn2a4+5/BJ/cqrRUtcwElxbvA+x/v1HVq3uEnT7Ncfc/gk/uUz+z5vtXk7PnrNVOjAh/grqP3PjCygTzktNUtk2fvP465q4jSN9iP5kf/PSo/LrGrS9tZrRmtM62SCDwlol7DLcpcX14mzYlclR5fl/coow9D2V23qwqTCiiiuwyCiiigAooooAKKKKACiiigAojkeOTej+W6UUUAXpI0v0eaJPLnT76f8AtRKo0RyPG+9H8t0q88aX6STQp5c6ffj/APZ0rD+EMo1Pb273D/8APNE+/JJ/BRb273D/APPNE+/JJ/BUlxcJ5fkw/u4P/Q6bbeiEJPOnl+TF+7g/9DqrRRWqVgCiiimAUUUUAFFFFABRRRQAUUUUAFFFFAE9pIkfmJKnyP8A8tP+WiUTwPBJ/wA9Ef7kn9+oKt2d4kcfkzfvIH/8crF3WqAqUVPcWb28n/PRH+5J/fqS3t08vzpv3cCf+P0/aK1wIo4H8l5vuJ/6HUNT3E7zyf8APNP4I/7lQVcb9QOa/wCFgaJ/z3n/AO/D10NncJeWsc8P3Jk8xK+nF/Y++EmOPDN7/wCDq+/+OV8izXFxpnxO8R+Hrd/L0bTNQvrG1t5Pn2QQTyJH8/33+SOvkMrz+GPqeytY7alD2Z0VFFFfYnEFNwadVTUNUstLjR7y6S3jf7nmVLkobjDU9Uh0e1+1XIeOPf5f7tPMrF/4WBon/Pab/vw9d98GfhRrnxK8c2sPi/w9qNz8Pbuzmvra+/49YJv9WbV0mjk8zOHr6A/4Y/8AhH/0K97/AODu+/8AjlfCY3iWnh63s4K/9f4jshhZ1D5M0/xhpeqXUdrbPNJI7+X/AKh6677ZCIP7P3/J/wA/X+f4K9y8WfsreAtG8F6/J4Z8NXY11LCafTwup3cjvcojyQ/I8nz/AD7Pkr5ehXxH4OGzx9Z3OgXVym+1j1KGODekf+sdPL/30rry/PKOYz9nPRhOhOmb89u8D7H+/UdW7e4S4h8m4/1f8En9yoLi3e3k2P8Afr6xN7M4iOiiitgCiiigAooooAKKKKACiiigAooooAKKKKACp7SN5Jt8T+Xs+fzP7lFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6xbb0Qy1eSJfwulp+72fO8f9/wD26yqI5Hjk3o/lulXpI0v45Jok8udPvx/+1Eqf4WnQCjRRRXQI6v4R/wDJUPDP/X1J/wCiHr37xL8JfAXjrUE1DxT4F8M+JtRSFIUutZ0iC6nRE/g3un3P8a8B+Ef/ACVDwz/19Sf+iHrxz/goX+1r8VPgH8bdI8PeCPFEWj6Nc6Ba38lu+k2V1++eSdHffNA7/wDLNO9fzvx/RqV84oxg7e5/7dI+xyecIYafP/McD/wVc+HPhH4d3nwuHhPwpoXhcX1tqD3S6Np0NoJ9jwCMyeX9/GT19avv+zz+3Wjvv8d+It/v4/T/AOP18g/Gr9pL4h/tEHRn8f64munSBMlkyafbWvl+Zs3j9xGm/OxOvpX7Q3n7YnwMlup5P+Fr+F/3j5/4+n/+Ir4av7bDUYQtzs7oclWcz4c/4Z7/AG6v+h78Q/8Ahfp/8fo/4Z7/AG6v+h78Q/8Ahfp/8fr7c/4bE+CH/RWPC3/gU/8A8RR/w2J8EP8AorHhb/wKf/4iuD6zX/59HV9Xo/8APw+I/wDhnv8Abq/6HrxD/wCF+n/x+mP+zz+3Xtfb478Qkf8AY/p/8fr7f/4bE+CH/RWPC3/gU/8A8RQ/7YnwMVcD4r+Fz/29P/8AEVf1iv8A8+x/V6P/AD8PSfGXgnwt8Qoo7XxL4c0fxNZQzedBBrNlHdIj/wB9EevE/jl8O/hl8JvAQ8Q6T4F8M+Gb77dBax3+jaLDBP8AvN/mJvSP7nArmv8Agop8cPG/7Pvwp8L614G1hdD1K91p7G6keygu98fkPJs/fxv/AB18MeDf2vviv8fNQPhjxx4pi1nRFV777Imk2Vr+/jB2PvgjR/437969fhnDV54/DVuf7ZzZlWh7GcD6W0/4gaJql9BZW083nzP5aeZA8ddJXh3gv/kcNF/67/8AtOSvca/q6jPnPg9Qooq1bwJt86X93An/AI/WzdjIsWGzyZPtf/Hj/wCP7/8AYqDVN/nx/c8j/lh5f+r2VHcXDzP/AM80/gj/ALlFnd+XH5M37yB//HK5vZu/tBkFFT3Fp9n/AOmiP9x/79QV1JqSEYNzba14deOfwbc3OgXz/JPPpN19lkeP/nm7/wC/X1b8DPj54c8W2fhfwTJqWpX3jWDSEj1D7VZznfNBBH50jzyD5/n/AI/46+b/ALtYUt/c/DG8/wCEn8MzSaX4gR3jS72JP8k/+s+R96V8XneSQxUPaUviO2hX9mffE3jjSV8S/wDCPNNN/a2/Z5fkP5e/y9/363+vy55rwL4Urqnjf4MaV49uPO1Lx/cGdzfqiJ5jJcyQIfJQeXnyU/uV3fhPxw+mWFxH401BLDVvO3pBdokcnkfwfc/4HX5nXwnu/ud46Nf3vK32Tkp5lOjiZ0Mw9z7UJ/Z5fs80pfaPQ+BiivH/AIjfFC7huLX/AIRfWIWj8l/O8iGN/n/5Z/fSvR/D/i3SfEDPBp+pw3dwiI7xpU1cFXo0oVqi3/rXsb4XPMFi8TPDQn78P/J/8JtUUVxPj7x5aaHpup2mn6rBBr0Kptg++8b4T/2TFc9OnOtP2cEenjMbRwNCdeueCftUfFb9m/wN8S7aw+L/AIf0nVfE8mkwTQ3F/wCG31F/svmThB5n130/9nn9pb9m61/tlvhrYw+H4ZLi1jvG0jwxNapK/wC88vfsHz/8tP8Avuvz5/4Kaa7f+Ivjvol1qNybi5Xw7bR+ZsRMoJpyPucdzXoP/BMCx+HN7ovj/wD4Tm5ht5Ev9L+yedeTQbvku9/3DXyGbUZ5fz1oz9+P83wntZNiKOZU6NTklyTX/bx4x+xF4++EPgHx54lu/jHplhqeiTaX5Nkl/ov9ooLrz0fIT+D5A4r7D/4aO/YZ/wChO8Nf+EI9flLLnznwO9d/p/wO8fanp9te2nhPVJ7S5hSeGdLYlJI3+44PvV4iNL46s+X52N6Uas3yUYcx+jv/AA0d+wz28H+G/wDwhHrzD4vQ+MviNHqnjr9k+/1bwh8H9EsJP7aXw5qY8PW39owxvPdSfZPPR3f7P9l/ebOQmP4K+BvEHh3VPCurz6Xq9lNYajbcTWs6bHT+LkV6F4A/ac+I/wALfh3r3gHwv4jXTvCesvcPqFh/Z9rP53nQJBN+8kjd03xoi/Iw6UU8Pye/SmROU/gmfYH7Bv7eul/DvS/G6/Gv4jeKNanuZbJtKGpSXWqlERJ/P2byfLzvj/IV+rVvIkkEc6fvI5kSRK/Ij/gmn+y/8Ofj5ovjm++I3hxtZGnXunR2Tfb57Xy0kS48z/UyJn7idf7le6/s2/tMfGP4eeKtQn/aj1e68JeEJNP8nRpNe0KCxSS9SaP93G8EAd/3O/r/APXr0KOJjz+yOSpC0Oc+8PEXiCy8M6f/AGhqUjR2e9E8yNN/36l0PWLbxBpVvqFg7yWk2/ZJInl/x7K8Q0v4kWnjzUY31TV4tR8FX++5sptnkpNHkPbPvRA9ez+FrfSLfQbRNBfzNJ+fyZEd5P8Alp8/3/8Abr6bE4P6tSg5r33/AOA2/wDkj5LLc1nmOKmocvJD/wAC5v8A5E1aMUtZOveLtJ8NyRpqmoQ2Dum9PP8A40rhjGUvdgfRVq0KMOevPkga1FJS1RoYXjLxjpnw+8K6j4h1qaaDSdPTfPJBC88n39n3E/36+N/jb8V9U+KHjOO98DeJNZXwc1hDZXNiZ57KCaTzJPP3wSD598ckf+/X2X4u8I6X498Mal4e121a70q/TZcwJM8G9N+/76fP/BXyB8bvg/rXw58ZCLwB4Z1WPwPFp8N/eXUf+lQQzeZJ5/zzyeZ/q0jr2sm+q/WV9Z/4ByYnn5P3ZyVnpdjpfmJZ2sNnG/8Ayzt49lWxVTT9YsdYjkewukvI0/5aR1bHFfttPk5PcPCCiiitgCiiigAooooAKkt7d7iTYn36Le3e4k2J9+pri4SBPs1v9z+OT+/WTb2QFv7ZD5P9n7/k/wCfr/P8FZ1xbvbybH+/UQNW7e4SdPs1x9z+CT+5Ucns9UMq0VJcW7wSbH+/Udb6MQUUUUwCiiigAooooAKKKKACiiigAooooAKKKnghed/+eaJ9+ST+Ck2kAWcbyTb4n8vZ8/mf3KvXkiX8Lpafu9nzvH/f/wBuqlxcJ5fkw/u4P/Q6qxyPHJvR/LdKw5Pae+MKKuyRpfq80SeXOn30/wDZ0qlW0ZcwgoooqgCiiigAooooAKKKKACiiigAooooAKKKKAPVfhj/AMibH/19T/8Aoysv4m7PtVh9s/48fsU/+/v8z+Ctr4VR/wDFExu/+r+1T/8AA/3lcx8ZJPM1LTf+vKf93/20r+HOGYX8T63/AF9r/wDtx6sv4J8n6p5/7h32fZPI/wBF8v8A1fl17jo//IF03/r1g/8ARdeDaPqCR2Mdrcp5lo6fPH/c/wCmkdfQFnZ/Z9J03Y/mQPawbJP7/wC7r+ysI7T5DzpklFFFeyZhRRRQAUUUUAFFVNQ1S10uON7ydLdJH8tPMqlb+C/ivq6HUNF8Laxf6Nc/v7K7gtoHSaB/9W6f9s683FY6hg/4rsaQp+0Niisj/hXXxs/6EjXP/AKCmRnV/CM0mn+NYZtE1l9k8FpfJGkjwP8Au43/AHf+2klc1DNcJiJ8lOoP2dQ2qKKK9syCiiigAooq1BAlunnXH3P4I/79S3YCxp+y0g33KeZA/wDyw/v1BqCP5nnb/MR/uSVBcXDzvvf79Ps7zyN6OnmQP99Kx5GvfGQUVPPafZ/nR/Mgf7klQVummroQUUUUwCiiigAooooAKKKKACiiigAooooAKKKkt7d55NiffpbAFvbvcSbE+/Wj9sh8n+z9/wAn/P1/n+CqlxcJAn2a3+5/HJ/fqoWrDk9pqxktxbvbybH+/UdWre4SdPs1x9z+CT+5UNxbvbybH+/VpvZiI6KKK1AKKKKACiiigAooooAKKKKACiiigAooooAKntI3km3xP5ez5/M/uUW9u9w//PNE+/JJ/BUlxcJ5fkw/u4P/AEOsW29EMtXkiX8Lpafu9nzvH/f/ANusqiOR45N6P5bpV6SNL+OSaJPLnT78f/tRKn+Fp0Ao0UUV0CCiiigAooooAKKKKACiiigAooooAKKKKACiirVvAm3zpf3cCf8Aj9S3YCxYbPJk+1/8eP8A4/v/ANioNU3+fH9zyP8Alh5f+r2VHcXDzP8A880/gj/uUWd35cfkzfvIH/8AHK5vZu/tBkFFT3Fp9n/6aI/3H/v1BXUndXQj75TNfnRr1ndWfxu8bXVzY3Vnaf21quy7ntXSB/8AS5PL/eP8lfb3w9+J7eONSvbVrCGz+zwxzCSCff8Ax1F8cfhQ3xn8Ex+HpNYbSAl7DffaPsv2r7m/93s3p/fr8NwWInlWLvVgelRxNHNMP7XDT54nw5/bmr/8JH5H2KT+xfM2fb/ssnkbP+u/3Pv10kciXCb0dJI/+mb19Sx/ALy/2dpPhR/wkMgV7V4P7W+xf37rzt/kb/8AgH36+TfiB4ZPwG8dJ4LW4TXvOhhvvt3k/Zf9fJJHs2fvPueXX3WVZ/DGV506it29DKthfZ6mjH+8p3wD8C2f7Q/iDWNL8S3V5ZQaTZx30EmjOkEju77Pn3o9Ot4/3lbn7AN5BcfEDxskM8Mkn9i2n+rfzP8AlvJXZxHWnTwj9m7GeFhzzPsbwv4btfCfhnRdBsHmktNLsoLGCSd98jxpHsTf/wB8Vr0mf4Fo2t/dr8e9T3XJLRsbk18bft26Xe3ni7wX9j069vP+JZfR/wCh2rz7P30f9xK+j/FHxOfw54zsdATT4ZxcfZf38l1sk+eTZ/q676ORoJfvuY9//fdddKpPBVIV3G/U4qWJo41zoUZ/B8R+cHiDWL23gg/sSD+1JP8AlpHaQPd7I/8AgFdBomqJfabaQ3zxx3fl/wDbSGT/AJaR7Pv19G/s/fszx/ATW9V1CPxM+uDULKG18v8As77KE2Pv37/Mf1rxj49fA0/CO61r4jprg1dtW1qTGmvZeR5P2qR3/wBfvff5f+5X6BR4ihiMUqb0h0OeeFnThznPT2728mx/9ZUdR+G9Y/tzQ7SS8jS38z7nz+Zs/eVYuLd7eTY/+sr72nU5zziOiiitgCiiigAooooAKKKKACiiigAqe3t3uH/55on35JP4KLe3e4f/AJ5on35JP4KkuLhPL8mH93B/6HWLb2QCTzp5fkxfJB/6HVWiitUuUAojkeOTej+W6UUUwL0kaX6PNEnlzp99P/aiVRojkeN96P5bpV540v0kmhTy50+/H/7OlYfwhm78Jf8AkqXhr/r6k/8ARD18cf8ABXX/AJOW8Pf9ijZf+lF1X2P8Jf8Akqfhr/r6k/8ARD18cf8ABXb/AJOV8Pf9ijZ/+lF1X4Vxp/yPaX/Xr/25n02W/wC5y/xHw+c0A09q+kf2L/2S4P2svEnifSp/E1x4aOkael4ktrp321590wTGzzE6Z9a+RnNQjzyOqPvHkdr8HfHt5Z29za+DPEVzaXCJPFPb6ZO8ciP9x0ITkH1FL/wpH4i/9CJ4n/8ABPdf/EV+hV//AMFMh+zbdSfCRPh5N4iHgEnwour/APCRvZi9+xf6L5/kCB/L3+Tv2b3+91NUf+Hz7f8ARI7n/wAK9/8A5Erh9piOkPxN+Sn/ADn50eIPCuueEb5bLW9Kv9HuWUSrb39s8MhQ/wAexwPSsPnj0r7l+N2of8NzQ3HxfuILjwhHpOlTaWmkpIdREhtUe43+e3l7N/nkbNn8Br4dZSrYPHNa4fFQxPNCPxw+I0q4WdGEJy+0frZ/wWG/5Ij4Q/7Gub/0levzp/Zr/wCSiv8A9g+f+Qr9Fv8AgsN/yRHwh/2NU3/pK9fnT+zX/wAlFf8A7B8/8hXq8K/x6H+I5MdtM+sPBf8AyOGi/wDXf/2nJXuNeHeC/wDkcNF/67/+05K94tIE2edL+7gT/wAfr+kcO7QufJTJLeBNnnS/u4E/8fqO4uHnk/55p/BH/couLh55P+eafwR/3KgrdJ7szCiiitgJ7O78tPJmTzIH+/HRPZ/Z/wCPzEf7kn9+oKns7zy/MSZPMgf76Vi01qgK7Vz/AI8jeTw++xHk/fx/6tK6i7tPs/zo/mQP9ySq+aU4qvT5Rn0R+y14o0PR/gJ4WtdQ1rTdPu4ze74Lq8SB0/06d/uO9dX4m0vwP4y1dL6fxVZ/amRLVI7TVbY7/n+T/wBDr4e1z4b2uuaxd6jczJ5k3/Tqkn/LOq9n8O7Lw3fWmto8NxJpc8epeRJapHv8iTzPL8z+D7lfm9ThyvTrTrUah0VnRxdH2FeHPA+75PgXooyhu9SLe8yf/EV5f4V17WvCN9Pc2NhI8kyeS/2qynf+Ou1/Z2+P1x8e4fEU0mgQ6GdMlhUfZb17rzhMjv8A884/7gr2KSGRfncTJ/10r5mOZV6PPQxS5/Js8fFcL4avOFfAz9jyHg3/AAvDxJH8jxaan/bq/wD8XWRYR3fjzxtaz6hDMiX82yaS1gdI/wDV/wAH3/8AnnXdal8Ev7Q1O9u21x4xcTPMI/sX3N77/wC/V7xZrM/wF+CepaskC62+jweekc8j2iT75/7/AM+z79d88bg6MF9Sh+8np9587RyPNsbX5MzqfuYe9/N/7d/KflH/AMFWfDlv4R/aE0LT7VpnifwtZzhp2y+TcXQ/pWF+w6c6b4o/7COnf+gXFUP+Chvxcn+M/wAbNI1uXSYdGlg8P21p9nhuftA+Wad9+/YnXf6Vf/YdXbpvij/sI6d/6BcV+Q8Xe0WAre2+LT/0o/fOF6VKji6MKPwW/wDbTyr9nv4X6L8UPEGuW2tyX0MFnZfak+wyIjlvOjjx86P/AM9K/Vz4a/sq+Err4b+EpRqGvKP7GsuPtMPH7hP+mNfkb8GPi5/wqXWNWvBpUeq/bbX7LskmMW394j5yP9yv3R+B91Lq/wAC/hxqrW7QJe+G9Lu9g/1a77WP5N9eLmuHrVK0va/w9OX1PVweIo0aMfZfGfmR8Sf2YbDxn/wUMl8A3tv4hXwRfX1raS6vbpiTJ06OTIm8ny87+PudPzrj/wBrf9k7SvgH8UtP0nwbp3ijxBpLaLbagY7yH7VJJcyPOHTzIII1REEaZH3+ff5P2fjnmjg2PM+z/nnvpbcX8iZg+07P+mO+uujmXseSnD7J5tbB83POcz86f+CW+vQ+BIfid/wnmoWvhSW/uNKkgTXJo9OM8aJdo+zztnyR/J9z6VjeD/G15+3hqy+Fvj+tr8OfDej276npt9o6jRJLm7d44djvemRH/cmR9keP6VT/AOCyn2hPFnwt8/zt39mXv+u6/wCvT/61eJ/tbftzTftTeCfDvh2fwfH4cOj3X2oXH9qvd+b+58naEaNNn96vXpw9t+/X2jzpz5PcPdvgn8etX1T4/wB38G7GLTdQ8HeG21DS9M1O3iEl7dWlmkqQPJNG+xt6RJvdE2Hrivs7w/8AFHWvCthBpsMVn9lh3/8AHxC+/wCd9/8Afr8xv+CXsZk/a40dUGD/AGRqeNvr9lev1c1b4ZP4g8S3E0uoTWkl06fu5LXfs/d/79fo+V42jUo+xxq91bH5NneRYqjX+vZM+Wf8vf8ArQz5/jN4kh2eXDpkkb/cf7K//wAcrR0fSrj4yfapvEaTWD2H7iE2MHkb0f7/AN/fVzS/gmdLvYJjrjSIkyP5f2L+4/8Av16qIpZRujRmFaYnFYSirYJe9/MTleVZpinfN5+7/J8XN/5MQ48uMJS9K8o/aH+N1z8BdF0O/j8Pw64dTvXtfLu7p7XZsgL7/wDVyf8APOu8+H+vT+OPA/hvxC1oLOTVtMtdQ+zp86Q+ciOY9/8AH9814XLLlVV/Cz9GUre4bZ7Vw/xc1zS7HwB4usLjVLC3v5tFvdlrPcok774HRPk+/Xj/AMWv2wL74YfEvX/Ccfgyy1SPTHgjS7m1h4Hk3wJP9zyP9uvCfH3iJP2hPF8HjW6sLbRbu1hg037JA/2v/USSSb97+X/z3/8AIde3l+U18ZOGlo9zkqVoUzA+F8ckelXaTJJHJ+4/1ieX/wAs67JaX/WUV+1UafsKageGFFFFdIgooooAKkt7d7iTYn36Le3e4k2J9+pri4SBPs1v9z+OT+/WTb2QBPOkCfZrf7n8cn9+qtFFWlYAoooqgLdvcJPH9luf9X/BJ/cqC4t3gk2P9+o6t286Tp9luf8AV/wSf3Kx/h6oCpRUlxbvBJsf79R1qncAooopgFFFFABRRRQAUUUUAFFFTwQPO/8AzzRPvySfwUm7K4CW9u8z8nYiffkk/gqS4nTZ5MI8uD/0Oi4uEZRDCPLg9f79cLe/EF7K+u4fskMvkyPH/r//ALXXyubZ5gclhCrj5cql/XQ+jyfh/Mc+lOllseacP8P/ALcdZI8aJ5jyeXHTPtdr/wA/UP8A33XB6p46TVLCeye3hj89PL/1++udkltlPzvDDX5FnvivhsHXjDLIe1h/29H/ANtP2HIvCTFYujKeaVPZT/7dl/6TI9hjvoI5N6XMccif7dW91rqvmPbvD9rj/ePBG/8A5ErxVWtJvuNE/wBK1vD+u/8ACN3z3VvGjybPL+/sry8v8XJYjEwp4qhyQf2veO/MfCJ4fDTqYPEc8/5eXl/9uPTO9DVJY3EfiDSYL6FEjndEd4I//ZKZX9IUK0MTTjWh1P51rUJ4ec6E/jgFFFFdhzBRRRQAUUUUAFFFFABRRRQAVagt08vzpf3cCf8Aj9LbwJs86X93An/j9R3Fw87/APPNP4I/7lY3cnZDPVPhzcPceEIn/wBX/pU+xI/4P3lcv8Yv+Qlpv/XjP/6Mrovhl/yJsf8A19T/APoyud+MX/IS03/rxn/9GV/EXCv/ACdCt/19r/8Atx6c/wCCfHtv/wAeqf7lfR2g3WzQ7CGZPMge1g3x/wDbOvnG3/49U/3K+hNH/wCQLpn/AF6wf+i6/s6gk5WZ58zSns/s/wDH5iP9yT+/UFT2d55fmJMnmQP99KLu0+z/ADo/mQP9ySu9Np2ZmQUUUVsAUyS4gt/9dOlv/wBdH8unMa83+Mn/AB6x/wDXjdVy16nsKbmM9N+Cfw5l+OXizUdD8VQato+nWdl9uguLG0e1dpPORNm+dHST5Hr7e8J+G7Xwf4V0Xw9YPNJY6TZQWMEl3+8keOOPy/n/ANur+izM3h/SdzMP9ChP3/8AYSrVfhWNx1THVXUqM9+nR9mN8uvFPjt+z/4e+IM194y1C91WDVNJ0h44YrWZI4H8jzp03o6b/vuf469tqjr2lDWtC1LS3fyPttrNa+fs37N6Onmf+P1x06s6M1UgzecVONj86vAfiS98UWM8955P/LP/AFCeX9+ukjngk+5Mkn/XN6p/GLwLD+yauj2kmuw+I/7Vsp7r/SIPsO37L5af33+/5lavxK+Crfs8+HtH8SRax/b41yf7ILWSy+y+T5kBn37/ADH3/wCrCfjX6/h86w8oUFz/ABngewmRUVU0u8/tDTbS6dPL85I5PLrWgt0t4/OuPufwR/36+n9oc4QW6W6edcfc/gj/AL9QXFw9xJvf79FxcPcSb3+/UdCT3YBRRRWoE9peeRvR08yB/vx0XFp9n+dH8yB/uSVBU9peeRvR08yB/vx1i01qgIKKnntfs/zo/mQP9ySoK1TTV0AUUUUwCiiigAooooAKKKKACiipLe3eeTYn36WwBb27zybE+/U9xcJBH9ltv9X/AByf36LidIE+y23+r/jk/v1UrL+JqwCiiitgCrUE6Tp9muPufwSf3Kq0VLXMBJcW728mx/v1HVq3uEnT7Ncfc/gk/uVDcW728mx/v1Cb2YEdFFFagFFFFABRRRQAUUUUAFFFFABU9vbvcP8A880T78kn8FFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6xbeyASedPL8mL5IP/Q6q0UVqlygFEcjxyb0fy3SiimBekjS/R5ok8udPvp/7USqNEcjxvvR/LdKvSRpfxyTRJ5c6ffj/wDaiVh/CGUaKKK3EFFFFABRRRQAUUUUAFFFFABRRVqCBNnnS/u4E/8AH6luwC28CbPOl/dwJ/4/UdxcPPJ/zzT+CP8AuUXFw88n/PNP4I/7lQVmk92AUUUVsBPZ3flp5MyeZA/346J7P7P/AB+Yj/ck/v1BU9neeX5iTJ5kD/fSsWmtUB9gfEnwTdeMrexhsJre0e2md38xH+f5P9iui8O6a+i+H9O0+aRJJLa1SB5I/wCPZHWiaOtfz3KvOpShSfwo9mnl9HD4ieMp/FMWvg79sqT7P+0RBO/+rj0XTn/8iT1941yviL4U+C/GOrf2jrvg/Q9a1Ly0g+16lYxzybE/1ab8f7b11YHFfU66q2OytD2kOQ+HvCPwi1r9paHVW8PahZ6GdJ2Ws/8Aa2+Tf53zps2f7lfoRFbwQt+5ghj3/I8kcKJvrF8L+BfC/ghbpfDPh7TNAS5dJLr+zbWODztn3N+ytyrx+PqY+p7SpsZUaPszyrxVrx+JV1d+E7O3ksbu1uXcXV0/yP5P+4d9N8K/CHVNB8TabqdzqFnJBbTeY8ce/wAx/wB29ei2XhfSdOv5L6302zgvpN++6jg+d9/360twpfXpwp+xoaQZ83TyGNat9cx756sPg/8AbTO1zTv7W0a9tU8mOe5tngSR0+58lcz8MfBF74Jj1JLy5hn+0ujp5G/+BP8Abrt+lFcka840p0b+7I9yWX0ZYiGMn8cQrwD9uD/kicZ/6jtj/wCz17/6Vl+JvCui+MtN/s7xBoun63Y70n+yalBHPHvT7knl0UK3sKsJ9jtnH2kD8/8Awf8A8ixYf7j/APoySult7hLiP7Lcfc/gk/uVp/Gb4NeKPA3jTxN4pstPs9H+GNlPHNDb2F0ieTD5cafJaoP+e/mfJXG2vgfxb420m+8aeHz5nhTTd/22SS98h08j55/3H8fyV+u4bOcJUwsJt2/zPDnQnzmxPbvbybH/ANZUdR+H/FFl4ohkRHfZD9y4kTy9m+rFxbvbybH/ANZX01Op7Q5yOiiithBRRRQAUUUUAFT29u9w/wDzzRPvySfwUW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAojkeOTej+W6UUUAdz8J40vPif4amTZHOk0m9P+2D/OlfFX/BXX/k5Tw/8A9ijZ/wDpRdV9j/CX/kqfhr/r6k/9EPXxx/wV1/5OV8P/APYo2f8A6UXVfgPGitnlL/r1/wC3M+ny/wD3Of8AiPiBf61+un/BXy6ns/gz4QmtZ3t5B4odPMhOz/l1k/uV+Ra/1r9cP+CxX/JE/CP/AGNT/wDpLJXwuK/jUfmepS+CZ5V8AvhZefsQ+G9J/aM8VaqnifQfEGhW8Y0nSY83qvqCRzpveb5Mps+c/wA66z4mft9eCP2oPAOvfCzQfD/iDSNX8UW32K2vdVW1+yxP5iTZcx/P/wAs65n4kfHLwb+0N+xz4L+D/gzVJrvxzpel6KLu0urZ7aCP7FabJ/37/JXzr8Pvgp4h+Dvi6y8Z+J/scWiaLJ593JaXUc7qmfLyqIfn+d0rycTiKUIT9pP999n/ANtPUweFq1Zwbh+6+1/7ccdffDm/+E3xi8IaJPqMV8JrqyugbUukeHmHHzj/AGK7/wDbc/4/PCP3fu3vRcf8tlrnPiJ470f4gfH7wPf6HczXdrDJp9q8k8PkHek3pX0J8XP2WPHf7S0thdeCoNNkj0N57e7+3aglr88j702b/wDYSvKU5rH4KeJ0nyyv62PalCEcHjIUPg5ontX/AAWG/wCSI+EP+xqm/wDSV6/Oj9mv/koj/wDYPn/kK/RT/gsJJ5/wL8GOn8Xih/8A0levz9/Zf09W8eSXd4HjsfsVzjH/AC2YbP3ae/Ir9B4V/j0P8R8RjtFM+qvh/p6SeJtJurmTy7RJ/wDvuTy5P3cde1XE73En/PNP4I4/4K8R8N6hJqHjTRnf93Gk/lpHH/q0j8uSvaq/pLDHyMwooor0DMKKKKACiiigCe0vPI3o6eZA/wB+Oi7tPs/zo/mQP9ySoKntLzyN6OnmQP8AfjrFpp3QEFFT3Vr5Hzo/mQP9ySoK1TTV0By/izwe/iCa0khnht44Ukj/AOWke/8A74ro/g343i/Zx8Ralrep2dxr0OqWUdjHBp02x08uTf5n781JUF1YWuoRol5apcRp9zzErxMbldDFwkmtzWE/Znr/APw3v4f/AOhI8Qf+BVr/APHK8V8W3R+J3xSvfiFaK9hY6nNHOmm3b75E8uCOD+D5Pvx76X/hF9H/AOgXZf8AfirdnZwafbxwW0CW8Cf8s468/BZBQwdb2qLnXnU0Ph39u7/kr2lf9gK2/wDQ5q6n9h4ltL8T5/6COnf+gXFct+3d/wAld0r/ALAVt/6HNXZ/sI6bPqOk+KfJT7moad/6BcV+Dca4PEY2eJoYaPPP/gn33DlajhsRSnWnyQPjub/Wv9a+gP2R/jFp/wCzb8WNN+IPiLTr3U7COzurWKxsZESaXzIfL3/P/BXj9jp8VhD/AGnqSlkY5trU/wDLxz3/ANisbUL+fVLt7m4ffI/U1jKHOuRnlyeraP1V/wCHv/w9f/mn/iX/AMDbavif9tT9o7SP2nvitpvivQtIvtEtLXRodNeC/mR5HeOSZ9/ycf8ALT9K+ejmkGawo4OhQnzwRtOtOfxDyzN3rtPhz8N7r4haheW1reWtk1rB57yXW/GN4XA2g+tReC/hrrfj23vp9Ighmjs3jjl8y5SPHmb9n3yP7hr6u0jwzpOiuJtO0mzsJ3gRHe0h2b6+4yfJp41+2raQX/kx5dat7NBo+kvpPh/S7BhCJ7O2htXmt08vfsTZJ+8+/X0x8H/2sNC+Ffw30bwpeeGNd1S60/z993aXUHlvvnkf+P5/46+eKn0/S31i+gsrZEknmfy08x9lfomJy2lioKmeRCo4O59m6H+3p4bjtZNngfxB9/8A5/bX/wCOV5/8WPidH+0VrGl3tlpt5otjpls9rc2mpTI/nb3Do6eQf4DGa4fwX4Lj0/R5INY06ykn895E+5J8nlx10lnpdlpfmfY7WG33/f8ALTZvqsHw7h6FT2wp4qczJ8P+B38L307vew6hBNB8n3/+en+3WFefDe6vL67n+2wx+dPJJ5e9676ivovqNPk5Dn9oZPh/R/7H0eC1m2XEieZ+8/7aVrRRpH9xKKK7KdNU4ciICiiithBRRRQAVJb273EmxPv0W9u9xJsT79TXFwkCfZrf7n8cn9+sm3sgCedIE+zW/wBz+OT+/VWiirSsAUUUVQBRRRQAUUUUAW7e4SeP7Lc/6v8Agk/uVBcW7wSbH+/UdW7edJ0+y3P+r/gk/uVj/D1QFSipLi3eCTY/36jrVO4BRRRTAKKKKACiip4IXnf/AJ5on35JP4KTdtwC3t3uH/55on35JP4KkuLhPL8mH93B/wCh0XFwnl+TD+7g/wDQ6qVjZy1YxBnJyc11Wl/DO61CxgvUurWOOZI5/Lkg/v1n+FbzS9PurttZhjuYXSPZHJB5/wA9dF/wuzwbpf8AoqT3sfk/J5cenP5aV/PniLxdmeT1YYbB05R/vuN4y937J97w9leExMfaYipZ/wAo7xdpT+H/AIKa3p7ukkkGnyfvI0/26qfs+uP+EDuyRGf+JpOP9X/0zjrgtU+JcGqfEnzptUvbjwTNMgmsZ0fyHj8j95H5H+/XsPgbVNB1jQZJ/DdrHZ6b58iPHHa+R+8/d+Z8n/fFfytjK1TE1Z4mt8c5c3/gR+nYP2fNGEH8HumN8Uvhne/Ei50X7Ff2un/YvPR/Pjf59+z+5/uV4n4H8B3XjzVL+ys72GzktU8x5LhHk3/vPLr234maH4y1ibSX8JajJp/k+Z9q8u98jf8A6vy/9/8A5aVxngvS7v4L31/q/izZb2moJHaQSWD/AGuR5/M8z7if7CPWFNvk0JxUFKtszBtb7+ydbk0A/vLvT3kt5J4/9W7pXQNFHexSTQ/u50+/H/7USvKfiBrn2jxBruqaVdTxx3V9JPBcR/u5PLeSr3gfxnayWNhpd5dXVxq0zyR+ZIjyf8tJPL+f/cr+2fDnNsRm2Te1xctYS5P/AAGMT8WzrDwo4rkgd3RRRX7CfPBRRRQAUUUUAFFFFABVu3gTZ50v7uBP/H6LeBNnnS/u4E/8fqO4uHnk/wCeafwR/wBysLuWiGE87zv/AM80/gj/ALlQUUVslbQD1X4Zf8ibH/19T/8Aoyud+MX/ACEtN/68Z/8A0ZXRfDL/AJE2P/r6n/8ARlc78Yv+Qlpv/XjP/wCjK/h7hX/k6Fb/AK+1/wD249Of8A+Pbf8A49U/3K+hNH/5Aumf9esH/ouvnu3/AOPVP9yvoTR/+QLpv/XrB/6Lr+zsP8Z58y3U9rd+RvR08yB/vx1BRXptJqzMye8s/s/zo/mQP9ySoOlZJ8f6Xpd1Pa3PnSJG/lzR+RUd54v0u38uRJJpIJPuSeRXD9ZpL3JzNfZmtcSeRDI//PNN9Z/hX4Q6v+01a6yNBvbPQ/7JgSCb+1t8m/z/ADNmzZ/uVe8I/Dfxh8XLy31nwmi3Hhu2vYLHUY7q8S137JI5J49j/fTyJK+5vDPgDwx4HmvW8NeG9J8Pi8ZBP/ZlmkAm2b9m/Z/vvXw+e59y/wCz4d6nZh8NzazKPiPxTB8PfD+lC4tpr/5I7T9w+z50j/2/9yuX/wCF72P/AEBr/wD7/JT/AAj4J16XVbv/AIS6NNW03Y/kx3Vz9qRH3/3P9yuw/wCEB8N9/D+mf+AqV8WvqlH3Ki5/NM8B/wBuZj++wtT2MP5JR97/ANJM2W4PxM+Hd19jT7BJfo8Cfav4Nj/7H+5XK6FrB+E1x/YGoQvqVxeTJdJNav8Au0R/k/j/ANyvUNN0+10uzjtbO1htLRPuQQJ5aJVfUPDOkateR3V3ptnd3UezZPPB5jpXPSxEI89KcfcPSxGW1qnJiYT/ANohHl5v/Si/cW8E0n76CG4/g/fwo9fNP7en/IgeDv8AsPP/AOkklfTFY3ibwX4e8bWkEHiPQtM1+1gk8+C31K1jnjST7m9N9ZYet9XrQqdj35Q56fIfC/hO3S38MabNc/8APBNif36tT3D3Em9vv1zurySW/wAa/FOlRSyLpdlq+owW1ir+XBDBG7xoiRj+BPLqzJcTeItYTwppHmf8JJqD/ZbL+CPf/rPv/wAFftmFx1N4VYmZ4Ps/+XZrUVkalpur/CXVE0Hxr+71m6RL6GOCf7VH5DybI/n/AN9HrXruwuKp4yn7SmZ8nswoooruEFFFFAE9rd+RvR08yB/vx0TwJbyfI/mI/wByoKKnl1uhhRRRVCCiiigAooooAKKKkt7d55NiffpbAFvbvPJsT79T3FwkEf2W2/1f8cn9+i4nSBPstt/q/wCOT+/VSsv4mrAKKKK2AKKKKACiiigAq1BOk6fZrj7n8En9yqtFS1zASXFu9vJsf79R1at7hJ0+zXH3P4JP7lQ3Fu9vJsf79Qm9mBHRRRWoBRRRQAUUUUAFT29u9w//ADzRPvySfwUW9u9w/wDzzRPvySfwVJcXCeX5MP7uD/0OsW3sgEnnTy/Ji+SD/wBDqrRRWqXKAUUUUwCiiigAojkeOTej+W6UUUAXZI0v0eaL93On34//AGdKpUUVMY8ugBRRRVAFFFFABRRRQAUUVaggTZ50v7uBP/H6luwC28CbPOl/dwJ/4/UdxcPPJ/zzT+CP+5RcXDzyf880/gj/ALlQVmk92AUUUVsAUUUUAFFFFAH33RSUtfzkfUBTqKbQAUUUUAFFFFABRRSUALRRRQBjeLvCWl+OPDd9oOu2v9oaVqEey6t/PdN6b9/30+f+CuO1X4U6P4K+Dfjbw74N0RrdNQ0/UJIbFJ3nee6mtnT5N7v9/wCSvSciilFyj1Hyn5t6ToupfC+3kh8Y6bc+FZ7w+dCmpJs85I/9Y6eX/vpW1ZWPirwiqR/ETT7/AEC0uf3enz6lDGm+T+OP93X1z8YP2dfDHxsutOu9cvtZs5LG2ntU/smdE3o7h33743ryr9vf/kXfAv8A2FLr/wBEV9xg8+qVKlDDx/vXPLnhfZ88zyvH/bSN/wB4kkf8dJmofCdwlx4Z0mG5f/l1j2T/ANz93Vq4t3gk2P8Afr9Ppz51aR5YyiiitxBU9vbvcP8A880T78kn8FFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6xbeyASedPL8mL5IP/Q6q0UVqlygFFFFMAooooAKKKKACiiigDq/hHz8UvDR/wCnqT/0Q9an7SHhH9l/XviBazfGqbQ4/Fy6dBHCdS1fULV/sm+Ty/kgdE/56Vx3h/XZ/Cuuafq9mkMl3ZyeYkc6fu/9Xs/9nrzz9oH4S6J+0t42tPFfi251K11G10+DTkj0KWOCDYkkj9Xjc7/3j1+QcVcOY7NswhisK/dUeX/yZn0GAxlGjRnCZ4R+2X8LfhC0nhZv2d7HT9Us44bptdfRb+6vtj708jzPPkfZ8m/pgcP6V9P/APBYIbvgn4QkH8Xih/8A0lkpP2af2Fvh5qOleIbN9X8VCOa9ggkxewf885P+mH+3XxF+0l+3F46/ac8J6V4c8U6P4d0+00y8N9HJottNHI8nl+X8/mTSZ4r86x2Bjha0MM53qw+P/wBtPTo1ueEpr4ZHjHgvx7rPw71d9S8P3Qs7xonhMjQpKdjDkfOpFdT4h/aE8d+JdFvdI1HWFn068Ty54VsrZN67w/3kQHqorzHjdzwaR/qa5KmEw9Sp7acIuXpqdEMViKcPZwn7p1Xwx3P8SPC2f+gpan/yMlfcHxS+NXxf+EEsEPws1CfT7a88+61NY9LtbsB0k2Jnzo5NnyYr4K0HWrjQdcsNUt1V7iynS4RW+6WRg4/DivtH4K+Kbj49aDrmseJ4LbTra0uY7WOLSldDLvSR5Ad7v6JXlYnD0I4+hi8b/Bhzc3zPYy+pKtg62Do/xZ/ofoh+03oPwX17w1psXxsm0yLw0moO+n/2lf3Nqn2ry/4HhdHPyV8j/FHw5+zBpfh+G6+Dd1oMnjNbiGFY9L1i9nk+xfP5/wAk8mz+5XN/tAfFLWP2nPDemaH4yjsbTTtOvft0H9h272z+Z5fl/vHkeT+CvIfCPwn0LwLrJ1HSptSefyHg/wBKmjePY/8A2zSv0TJeFcdg8TRxFSekZHy2Jx1KtCaiel+C/wDkcNF/67/+05K9xrw7wX/yOGi/9d//AGnJXuNfumG+A+ZmFFFFdhmFFFFABRRRQAUUUUAT2l55G9HTzIH+/HRd2n2f50fzIH+5JUFT2l55G9HTzIH+/HWLTTugIKKnurXyPnR/Mgf7klQVqmmroAooopgfI37XHwh8afEL4mWWoeG/DGpaxZQ6TbwPPaQF0D75Pkr0b9iP4aeIfhhpfih/F+hXuiPeXti9pBdx+X5+xJ/M/wCAfvEr323gS3X7TcplP4E/v0z7Q9xdI7/6zfXxkcggse8Zznb9a9zkPy8v/hR4+1e6kuZvDepyyOevkGq3/CmfHH/Qran/AN+DX2/4D8L2Xii+u4Lx5o44YPMT7O//AE0rF1S3Sz1W/tU/1cM7xp5n+xJXgPhTDX+OR1/XZ9j47PwZ8bt18L6n/wB+DXrXwv8Agxpv/CNyf8JX4clXVheSbftMs0LeTsTZwjf399ev0V1YbhzDYWfP8f8AiInipz2Mjw14Q0fwfHdR6NZ/YIrkpJMPOd9+zfs++/8AtvWvRU/9n3v/AD5XX/gLJX1FOnCnD2dM4229WGn6fdapdfZbOB7yf/WeXH/rK9U8GeC7Wz02wvbzT3t9aheST95I/wAn7yT/AJZ+Zs+5R4M8F2Wlx6bq6Pdfa3tfnjk/1fzx/vPk8uuv+9XpUaP2pGXOLRRRXcZhRRRQAUUUUAFFFFABUlvbvcSbE+/Rb273EmxPv1NcXCQJ9mt/ufxyf36ybeyAJ50gT7Nb/c/jk/v1Vooq0rAFFFFUAUUUUAFFFFABRRRQAUUUUAW7e4SeP7Lc/wCr/gk/uVBcW7wSbH+/UdW7edJ0+y3P+r/gk/uVj/D1QFSipLi3eCTY/wB+o61TuAUUVPBA87/880T78kn8FDdtWAW9u9w//PNE+/JJ/BUlxcJ5fkw/u4P/AEOi4uE8vyYf3cH/AKHVSskm9WMKKKK2ENyOmeaXw94b8N3Gv79cto/sjpJI++aSP5/+AVHcXEFv/rXjjH/TR65W88YXtvdTxwpDJHG7xpJsr8q43zDI4YP6rm7+P+7GUo/3v7p9FlMMV7bnwx6vceGvhTbq8ky2ccafxyXt1/8AF1r+H/Gfw58KWEllpetafZ2jvJP5fnyP8/8AwOvAtQ8WXt5YzwOkPlun/LNKxY99fx5nWGymnXhDKZznD+8fsWXUswq4KeNnCPuT5fdPqWT4s+DE258S2AHfBf8A+IrzL9oDxppfiTwrpMHh7V7bULuHUfMnjg/gj8iT+/XkV1nIx61a5rzcvrQwOKhieSM+T7EvhPXzbCOOVYepz+/W5v8AyWRy15qF7J5lrcyfx/PHsqfwneQaf4m0m6uXS3ghn8x5JP4Kn1jS08iSdN8k7vWNJG8cnzp5df2zwlmmWZlgOfL4Qh/PCEeX3vd5vdPwHMKNehWvWPfNP1Sy1iCSewukvIEfy/Mj/v1aXpXG/Cf/AJFif/r+k/8ARcddnX6ZTlzwPDCiiitRBRRRQAVbt4E2edL+7gT/AMfot4E2edL+7gT/AMfqO4uHnk/55p/BH/crC7lohhPO87/880/gj/uVBRRWyVtBBRRRTA9V+GX/ACJsf/X1P/6MrnfjF/yEtN/68Z//AEZXRfDL/kTY/wDr6n/9GVzvxi/5CWm/9eM//oyv4d4V/wCToVv+vtf/ANuPUn/APj23/wCPVP8Acr6E0f8A5Aum/wDXrB/6Lr57t/8Aj1T/AHK+hNL/AHeh2G//AJ9YP/Rdf2dh/jPPmWelY5XWvGc0dl4Hhm1/Uof391aaakbyJB9zzPn/ANuqPijXNbt4PP8ADGnTa55MEjv9gsZ77Y//ACz8zyPuV9lfCP8AZx8MfCHXLrW9HvdZvLy/svssyalMkkcaO6P9xI0+feleFnedQwcPZ0/jOmhR9ocx4D/ZV8Bar4H8PX3irwrcf8JPd6dBPqwn1G8gcXTx/v8Aekc+xPn/ALldRH+yv8KbeB4E8Jny5v8AWR/2ve//AB+vVsCjivyieLxE3d1Jfez2OSBzfgP4aeGvhfpNxpvhbT/7Msbq6+1PH9qmn3zbETf87v8AwJH+VdNTo6PLrnu5fEWNoooqwCiinUAFNoooA+cfj9+z/ptt4Z1jxP4E8LXlx46vNTSd5LG5mmkfz5x9rk8h5PL/ANXmrn7PfwB0mLw14d8V+LvDNzafEa2up53murmZJEdJ5I4P3CSeX/qNlfQOM0YxXV9cr+x+r87t/X4eRHsYc/tD4d/bR/5Lpof/AGAbH/0ru65jUNcstHf/AEy6S33/ALxPMr6h+O37Pfh/4iTXfi++utZj1nTNIeO1t7GePyJPJM00fybHd/nkrwf4C/BIfH6PWpPHVv4g0CTThara/ZLV7HzPO37x+/R9+zYn3K+0ynOKODwM5S3h+rPLr0J1KmhixnfHvSiuT8H+JL3WNSu7WbyfItUk2eWn7z5JPLrqxX6LQrwxEPaQPPFooorqEFFFFABRRRQAUUUUAFFFSW9u88mxPv0tgC3t3nk2J9+p7i4SCP7Lbf6v+OT+/RcTpAn2W2/1f8cn9+qlZfxNWAUUUVsAUUUUAFFFFABRRRQAUUUUAFWoJ0nT7Ncfc/gk/uVVoqWuYCS4t3t5Nj/fqOrVvcJOn2a4+5/BJ/cqG4t3t5Nj/fqE3swI6KKK1AKnt7d7h/8Anmiffkk/got7d7h/+eaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MXyQf8AodVaKK1S5QCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVaggTZ50v7uBP/AB+pbsAtvAmzzpf3cCf+P1HcXDzyf880/gj/ALlFxcPPJ/zzT+CP+5UFZpPdgFFFFbAFFFFABRRRQAUUUUAffgQnooP40tcF8O/h/deCb6+nmubaeOaFI/3e/wAzfv8A9uu66V/OdWEKc3CE+Y9nBVq2Io+0rQ5J/wAotFFJSO7yClrjvHXxEtvA9xaxz2Fxd+cjv+4dE+5/v12GcrkUSpzhCM5rSRyUcVQxFadCnP34fGFLSCsrxNrqeG9FvtTmhknS2j8zy4/9Y9TGPNLkidE6lOhCdSp8EDWKsvVaThq8hh0Wf4peILTxPYyJY21rNBDJDdb3d/Ik3/wHZ/HXsDLtfrW1ek6Noc3vfkeZl+YTx3PPk9z7H94SiikrI9YWiiimMkhiaR48o/l7/wDWV+fnjz41X37TllpVlf6TZ+HI9Jd75Hsbp7vzt6bPLffHHsr6D/aW/Z11744654bu9I1rTNLj0+1ntZEvvO3u7yRyfJs/6518veFPFdr4kkf7NZSW/lwpJ+82V9fw3g8NWqe0nP31sv8AwI8zFTn8Bt6fZ/2fptpa7/M8lI4/MrSt7hJ4/stz/q/4JP7lVKK/WPZ6HlElxbvA+x/v0+3t3nb/AJ5on35JP4Kn0/8A4mCfZpvuJ9yf/njRqEn2f/RkTy4E/wDH/wDbqfaO/J1GR3E6eX5MX7uD/wBDqrRRW6XKIKKKKYBRRRQAUUUUAFFFFABRRRQAntU+lWQ1DVNNsN/l/arqC08yP+DfJsqDg811/h3wbKnhG7+IJukGmeH5JtRnsY4/386WR89443+5+8RK8LNMfRy7Czr1Z8v/AMkdNGjOvPkiY/x8/aGH7ANx4et7fQD46TxM8900l1efYfsxtfLjA+RH3/6yvxymYs2a+tP27P2tfDf7Vl54Kk8N6Lq2j/2HFepONV8n5/OeN02eX/uH865yT9ivVY3dD4r0klP+mU//AMbr+Y8VmdOjP6zj52nM+6w+X1sUuTCw92B81UV9Kf8ADF2pf9DZpH/fqf8A+N1dsf2JdQMJubvxbpMdov8AcinLuf7n+r61x/29lm/tjs/sPH/yHz1pOkw3ELX+oM8enxts/d/fmf8AuJXqPwp/aKl+HNrq1qdCh1BL6eGSNftMkK2yIkibEx/10rtNS/ZF1vWJkeTxdoqRqu2KCOGfZEn+x8leWfGL4MXHwjl0sT6ra6l9uR5E+zK67NpxzvAqZY7Lc0/2Tn5+b/EOGCx+Wv6y4cvKfbul/DNNQvp4P7Rn+T/nnB/9sriNUjg0/Vb+y89P9Fnkg/ef7ElQeE/itpvx0vL230SyvNMWwT7U8l86fPvcJ/B9a9a8H+JrO8uLTw99gk8+1tdjzybPLfZX7PwTnFbMsH9Wx9X/AGmHxen2dvdPns/wlHDVva4SH7l/CefeC7hJPGOi7HT/AF//ALTkr3Q037Okf3ET/vin1+r0Ych8gFFFFdIBRRRQAUUUUAFFFFABRRRQBPaXnkb0dPMgf78dF3Z+X+/hfzIH+5JUFT2l39n8z5PMR/vp/frFpp3QEFW7e3S3Tzrj7n8Ef9+pvsENnH9tf95av/qY/wC//v1SuLh55N7/AH6SftdhhcXD3Em9/v1HH+7k30UVtbSwjm/CfgdPCd1PMmoPeecnl+XJB5f/AC08ysy8+FcF5fXd0+qTRyTTyT+X5CfJvk/66V29FYexiHOcD/wqO1/6C83/AICx/wDxdH/Co7X/AKC83/gLH/8AF13vNHNT9Wga8559J8H7WSCRP7Xm+f8A6dU/+OV6FGX2ffeSik2itIU4w2MxaKKK2EFFFFABRRRQAUUUUAFSW9u9xJsT79FvbvcSbE+/U1xcJAn2a3+5/HJ/frJt7IAnnSBPs1v9z+OT+/VWiirSsAUUUVQBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAFu3uEnj+y3P8Aq/4JP7lQXFu8Emx/v1HV7T/+Jgn2aX7ifcn/AOeNYP8Ad6gQW9u87/8APNE+/JJ/BUlxcJ5fkw/u4P8A0On38n2f/RkTy4E/8f8A9uqNC/easYUUUVuIKKKKALeleA4PiBPJazX0+npap5nmQRxvv/77rN+IfwfsvAvha41dNaurzy54IPLngRI/nfZXdfCv/kMan/1wT/0ZXT+NvFtv4H0CXVbm1mu4VmjhCQhN/wA/+/X8NeJ0pf60Vl093/0mJ+2cK4FYzLYQh8c//kj5h8H6OvizxZpWjNcNFb383k/aIU3un7t/4Kv/ABK8Fx+APEkWlR3Mt5FJaJc+ZdQ7H+d3T/2Sq/jrxND4x8X6jrdpFNZrc7NiM/zpsjRP4P8AcrA9nd5P+uj76/L/ACSP6dyTKHl0abg+Rcvvw/vfzHoHwm+E9v8AEi31Iyald6cLKSGL/R7dJvM37/8A4ivPluhE0hldIwknl/vKUSSRj5HeM/8ATN9ldn8LfHVp8P8AWr+9ubS4v47m2SFY7XZ8nzl/46WmzOLPsjnmFGtVfvy05Evsfzf4ubzO08N/AOy1/wAO6Tqb+Ib2P7bawXflx2qfJvj31x/iz4JwWfiS+g/te6kRNn7yS1T/AJ5/9dK+ktB1Vda0PTtQiSSGC8tlnRH++iOm+vlj4yagmj/H7UtRdHkS1e1keOP+P/RI6/ZvCOUv7crKf/PqX/pUT+ZOLcPTw2ChDrzf/JHSeF/DieF9NkskunuEefz/ADJE8v8A55//AButYVk+F/EcfijTpL2G1e3RJ/I8uT/P+3WsOK/sqny8vuH4+FFFFbCCrdvAmzzpf3cCf+P0W8CbPOl/dwJ/4/UdxcPPJ/zzT+CP+5WDblohhPO87/8APNP4I/7lQUUVslbRCCiiimAUUUUAeq/DL/kTY/8Ar6n/APRlc78Yv+Qlpv8A14z/APoyui+GX/Imx/8AX1P/AOjK534xf8hLTf8Arxn/APRlfw5wr/ydCt/19r/+3HqT/gHx7b/8eqf7le+2/wDyKkH/AGDo/wD0RXgVv/x6p/uV9CaP/wAgXTP+vWD/ANF1/Z+HPPmeo/8ABPrd5PxD+z7vv6d/qP8ArnPX1p1r82fHHgufxbBGltPDZ/uJIP3nmfx/7lfX3wQ/aS0v4xeIr7w5YeHtS0eewsFu/Pu5oXjdEkSHy/kr8tz3LK2HxDxD1T/4CPVwtb3OQ9oo6Vj+LvEkPg/RJNTlhedEdE2Rvs+/JT/CviJPFmgw6nFC8CTO/wC7k/1nyPsr5n2c+X2v2djT65R+s/Vef3+Xm/7dNanLTaTNSdY+m06jy/8AlpUgFMrO8Ra1H4e0W91KaF50to97xx/x1Q8FeLIfGmnz3kVtNaJDN5Hlzuj/AMG//wBqVfs58vtfsnHPGUIYn6tz++dDRRRQdgUUUUAJ/q6s2rSzTQv8zoj1k61qkeg6NqWqTI88dlazXTxp999ib/8A2SvhD4y/Eq3/AGnrjQL7SbC50CPRraeB/wC0pkfzvP8ALePZ5B/6Z12YPA1MfU9lTMJ1vZnn/wAN5E/tzVk3/vP3/wD6Pr0KmW9ulvHGmxPkTy/3aU+v3TC0Pq9PkPngooortAKKKKACiiigAooqS3t3nk2J9+lsAW9u88mxPv1PcXCQR/Zbb/V/xyf36LidIE+y23+r/jk/v1UrL+JqwCiiitgCiiigAooooAKKKKACiiigAooooAKKKKACrUE6Tp9muPufwSf3Kq0VMlzASXFu8D7H+/T7e3edv+eaJ9+ST+Cp9P8A+Jgn2ab7ifcn/wCeNGoSfZ/9GRPLgT/x/wD26x9o78nUZHcTp5fkxfu4P/Q6q0UVulyiCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVaggTZ50v7uBP/H6luwC28CbPOl/dwJ/4/UdxcPPJ/zzT+CP+5RcXDzyf880/gj/ALlQVmk92AUUUVsAUUUUAFFFFABRRRQAUUUUAfQPgH4hHQtSvZtb1DUru3eFEhjkd59j7/8Afruf+F2eG/8AqIf+A3/2dYfxC+GCWtrYnw1o80k3nP52yd3+TZ/tvXE/8K38Vn/mBXv/AJD/APjlfkPscuxv76U+Ty0Pz767xBkf+xwhz8nXllI9E1f4j2XjDTZ9G0Rr+31i8Gy1d0MPz43/AH/4PuVs+GdC8Q2PgfVbK/unk1mbzvss/wBtd9m9Pk+f+D56q/D/AOHVjYabpWo3+mvBr0Rkkd5J3/dvvf8Ag37PuV2uqaraaDpN/qF9MlvY2UD3V1PJ/BGib3k/74r57E1KMX7HDbLv/Wx93luDxuKX1zMJ+/OH2P5f73948Tvvhf4yvlU38sN28fyI9zqRkKf991H4B+IzaDfXk2tX+pXdtNFHHFHvebY2/wD369O8B/Fbwp8UIb6XwprtrraWEiR3XkI6bN/+r++lNj+EPhKP/mE4/wC3qb/4uvRWZRqU5UMZT9LKzX3niVeF6+ErQxWU1vf+3zy/+RiZn/C7PDgHH9of+A3/ANnXnWu65f8Ajnxdd22nX939j1KXZDbXEzon+r/uf8ApNW+G/iL+1r37HoVz9j+0v5OCn3N/yfx12Wh6L4b+Hvg2Hxb4ug/sWXTU866urt3/AHHz7Eyif76V2f7FgYKtRfPN7Hif8LWf1/qWKhyQh/dlE4XbrXgLXrGzvLy5tIY5obuaCyuXdHTf/wDYV6lJ8bPDm7mO/wA/9e3/ANnVGL/hC/jJ4d1HxL4ZdPEM8KSWUNzC80f79E+5sfZ/fT8684/4Vx4r3ceH7z/xz/4urp/Usy1xXuTj/wBuirQzfhuc6GChzwl/ikepn42eHmH3L8f9u3/2dV/A+l+IrjxXLrU95NJoF4k09vBJeSPlH+eP9z/BWB8O/hfPdaje/wDCS6NNFaeTH5PmPs+ff/sPXsFjYwadYwWtsnlwWyJAkf8AcRK8fGSw2HlKlhdfP/I+oyiOZ5qoYrMfc5Pgh8Mv+3ieiuLf40eCV8ff8IQ3iS1/4Snzvsv9l7Jt/neX5mz7mz/V11Wqata6HpV/ql/MlvY2UD3c9xJ/AiR75H/74rxvePvS4Mx9a+Pv2yvB+g/D/Q/B03hXRdN8MSXV7NBO+i2sdpJNGkH+rfy/v17VD+1d8JNyO3jnT9n/AFxm/wDiK+QZvFHjf4pQwxfEe8utXhs/39sbiCCDZO/yP/qI0/gr6XJMBiquLU17lvU5MTOHIWfD8jyeH9Nd3eSR7WP95J/1zrWt7d52/wCeaJ9+ST+CoNL0uOO3jtYf3cEKeX+8/gSrdxcJ5fkw/JB/6HX7J/cR4YXFwnl+TD+7g/8AQ6Le4SeP7Lc/6v8Agk/uVUop+zVhElxbvbybH+/UdWre4SdPs1x9z+CT+5UNxbvbybH+/TTezAjooorUAooooAKKKKACiiigAooooAvaHod14k1y00qw8n7XdPsj+0P5cf8Aq9//ALJUvxY/aE8Ifs2/DvxH8L/G15eweLNb0W+urRNNtftVu8d1BPBD+8+T+NK2PhH/AMlS8Nn/AKepP/RElfHv/BXJtv7Sfh4+nhKyP/kxdV+JcbYyrUxsMsf8Pl5/+3vePo8tgvYzrfbOE/ZF8N6LrnhbxTNqekabqbx3tqiPe2aTumUk4Tf9K+s/Avw6134k32o2uhfY/tFnCk8326Yw/I77P+edfK/7Hchj8E+M5F/5/bT/ANFz1+r8Wg/Dv4FGfVNln4XGoMLL7RPPM/n7Pn8v599fzRmOHeMzOt7WfuQ/+RP13A4v6nllFU4e/P8A+SLHhP4U+GrDwvo9tqvhXQLi/t7OGG6kOnwyb50j+f59nz/P/HWm3w58KFkL+E9BMafczpkPyf8AjleR/Cv4ua/40+N2r6VJrKah4ZRNQn0+OO1gSPYkkfkfPs3/AHDXp/xc1y98N/DHxLq2lXP2PUrOz8+G42I+x96fwPXTCVCdHngebOFanWUKm8v1PH/jX8CdT1TxE+reGbDQ9M0G20xPOgR0tf3iGR3fYkf9zy6+bm8P6P4kurGTVtH03VkT7n9o2ST7I3/36+rvhL8Z9N8R+E5NM8c+JLObXdQvZrEWs6iF5oX2IiYhT+Pe351wPx8+H/h/4d+IPB1r4f01NNjuVneaNJ3ffseNI/vu9eHiqMYpYvDvU+iweInCTwOIWh+aP7NvxQ0D4dax4hfxBNcQQ3lssMBtbcTHeJg/9K+p/COpQ6bqlr4qnEg0e8tfPheP/WbJ03p+7r875OGds8EkV996Od3wi8JH10zTv/SVK/bspxUsqzmhUof8xE4wl6H546f17A14Vv8AlzHmie2aPqkGuabHe22/yJvueYnlyVaHSuc+Hf8AyJem/wDbT/0ZJXRjpX9SQ+A/MhaKKK1AKKKKACiiigAooooAKKKKACrdvbpbp51x9z+CP+/RBbpbx+dcfc/gj/v1BcXDzyb3+/WP8TRATx6g/nyO/wC8jf78dR3Fn5cfnQv5kD/+OVBU9pePbyf89Ef76f36HC2sBkFFWprVI086L54X/wDHKq1pF8yEFFFFUAUUUUAFFFFABRRRQAUUUUAFFFFABUlvbvcSbE+/Rb273EmxPv1NcXCQJ9mt/ufxyf36ybeyAJ50gT7Nb/c/jk/v1Vooq0rAFFFFUAUUUUAFFFFACD5aXmtrwFY6ZqPjnQLbWYkfSJrnZcxyO6fJ5cn9z/tnX0p/wrn4Jf8APna/+B1x/wDF18TnHFeW5FWhQxsvel/h/WSPUw2AxOKhz0YHyhzRzX1d/wAK5+B//Pla/wDgfcf/ABdH/Cufgf8A8+dr/wCB9x/8XXh/8RGyT+b8Yf8AyR1f2Lj/AOT8H/kfKPNHNfV3/Cufgf8A8+dr/wCB9x/8XR/wrn4H/wDPna/+B9x/8XR/xEbJP5vxj/8AJB/YuP8A5Pwf+R8o80c19Xf8K5+B/wDz52v/AIH3H/xdH/Cufgf/AM+dr/4H3H/xdH/ERsk/m/GP/wAkH9i4/wDk/B/5HyjzRzX1d/wrn4H/APPna/8Agfcf/F0R/DX4JSfcsLX5P+n64/8Ai6P+IjZJ/N+Mf/kg/sXH/wAn4P8AyPlm3tJLh/8AnmiffeT+Cnzzr5fkxfJB/wCh19T/APCv/gr5Pk/ZLXy/v4+3XH/xdR/8K5+B/wDz52v/AIH3H/xdY/8AERckbu5/+k//ACQf2Lj/AOT8H/kfLkFwtwn2a4/1f8En9yobi3e3k2Onz19U/wDCufgf/wA+dr/4H3H/AMXUknw/+C0iIj2lr5afc/064/8Ai6F4i5In8X4x/wDkg/sXH/yfg/8AI+TeaOa+rpPhr8EkbY9haj/t+uP/AIuj/hXPwP8A+fO1/wDA+4/+Lrb/AIiNkn834x/+SD+xcf8Ayfg/8j5R5o5r6u/4Vz8D/wDnztf/AAPuP/i6P+Fc/A//AJ87X/wPuP8A4uj/AIiNkn834x/+SD+xcf8Ayfg/8j5Ytry502R2tp5reR/vyQSbK5Txp4wn1jTZ9Oe9vbjy503xzu/lvskr6A/aA+Hvhuz8MaVJ8MtPW41b+0B9pNrdM7/ZfIn6edJs/wBZ5dfPknwv8aySSO/h668x/wB4/wC/g/8AjlfjvHHG2X46n9Wy+muefxTlGP8A5LLmPs+HsDjcurQxX8nw/EcRJJJGjyOn7tKwdQ1B7iffDJNHHs/6517j8Ovg/fXnjzQ4fF3h5/8AhFZJvL1P7ROiR+T5cn9yTf8A6zy6+kn/AGcv2cFUEaHZZ751O9/+OV85wPicjwPPjMw+P/t3l5f+3j7LiXiXP80pfVeW0P7sZcx8AaXqn2fzPtLzSf3P+Wlb37z+5X3H/wAM5fs3MuRoNn/4Mr3/AOOV8lyfCnxh5lwE8OXATzJPLHnwfc3/ALv/AJaV53G9bJsViYY3K/jn8fw8v2eXl5Tfh7ijP8vwv1OpH3I/3Zcxxlx4o8X6fa/ufFGr29pD+7gt7fUXjjSP/lnHsrmtQ1C91i6kur+6m1C7f79xdv5kj/8AbSvVbj4R+LriPZN4bupI/wDrvB/8XX0J8H/2efhDefDrSpfHnh2GPxVJ5/2xZ9QuUb/Xvszsk2fc2V9/wfx1ldKh7HG0+StD7cYxj7vu/wB74j85zjK8biK068ft/wCI+LtL1TUbOe0gtr26t4Hnj3x287xx/wCsr3+WMxzSf79fQsf7O37OUciOmgWgdG3g/wBq3nX/AL+V0v8Awrn4JP8A8uVqf+3+5/8Ai6/UKXiHkkPt/wDpH/yR85/YuP8A5PwZ8o81bt7NfL86b93An/j9fUf/AArT4JeXv+wWvl/9f1x/8XT5Ph/8Fptm+0tTs+RP9OuP/i61n4i5J/P+MP8A5Ij+xcf/ACfg/wDI+Vri4knf/nmn8Ef9yq/NfV3/AArn4H/8+dr/AOB9x/8AF0f8K5+B/wDz52v/AIH3H/xdNeIuQr7X/pH/AMkH9i4/+T8H/kfKPNHNfV3/AArn4H/8+dr/AOB9x/8AF0f8K5+B/wDz52v/AIH3H/xdP/iIuSfzfjH/AOSD+xcf/J+D/wAj5R6cUe9ani63tLPxbr9rpvl/2fBezR23lPvHk7/krLPSv0jD4iGIpQrw2keHUp+zfsxaKKK6iT1X4Zf8ibH/ANfU/wD6MrnfjF/yEtN/68Z//RldF8Mv+RNj/wCvqf8A9GVzvxi/5CWm/wDXjP8A+jK/h3hX/k6Fb/r7X/8Abj1J/wAA+Pbf/j1T/cr6E0f/AJAumf8AXrB/6Lr57t/+PVP9yvoTR/8AkC6b/wBesH/ouv7Ow/xnnzLRrGvLLU9JEc3hO5m0C+f5J59NunsZHT/nnvT+DfW1SYrqxFCGIhyVCD3v4M/HXwr4ysPDPw61ddS1fxVFpscV7Jqdt58Et1axh53ed3+c5/j966zVtH1bwj4qbWI5Psvg+zdJ3srW52J5ez5/3P8Av18l/BfVrTwH8eLTxP4lYaV4egN6r31wn7tDMjpH9z+/X1Lqn7SXwb1vT57K78Z6fcWs6bJE2XSb/wDxyvx/GYWeBxXs6dOThLy/rU0xeDhjqPtOfknD4P8A7b+6b1/8adCudNu4IRfRzvC8aSfZtnz7P9+uS+HfxKTw1b6idevNS1APs8n53n2bN+/7710XgXSvhf8AEjTLjUvDIh1eyhm+yvPBPdR7J9iPs+f/AGHj/OqHjz4VpazWTaBo89xE6SGdEnd9z/wffeijLLnz4bklFy/mtpY+Rx9HiLDuGYOcJqH8nN9r+6evRyLJBG+fkdPMSs/xD4ltvC2lvf33neQjon7hN8nz14MPil4ri+RNakj2fJ5fkwf/ABFaN3H4/wDGWkotxbXmoafcbJk+SBN39x/5Vi8mqU5/v6kVD1OqXGVKvRnDB0Zzn/h/+2E1nxgmpeNo9Re5vJ/C00yedYyb9jx+X86SQ/c+/XqvgbVNE1bTZ5tCs/7PtUm2PH9l8je+xK4zTvhuY/h3cXM2iTf8JNsf/R/Pfe/7z5Pk37PuVx8mp+Jvh/H9jhmm02Cb9/5EiI8n9z/2SuudChjo+xw0rOHu69fPS54mFxuNyOt9azGHPCt7/wAPwc32fe5eX/CfRHBrE8UeLdO8H2sFzfedsmfyE8hN9eIf8LU8UxR/8hqT/vzD/wDEV2fhjQ/EPjG/aHxrZ3E+kpD59v8AaNkKed/wD/Y3159XK54O08VJcvk9fkfSUOKaeZP2GXwnz/3o+7/29yyPTtJ1KDWdLtdQtt/kXMKTp5ibJNj1ZqtptjBpdjBZWyeXa2ybEj3/AHEqzXiy+L3T7qnz8nv/ABkVxbwXkE8FzClxBMjo8cib43T/AJ518zftCfs06vr994cX4W6FoegWcME6ajHZzJpfnO7p5H+rj+fZ+8r6Yvr6DTbG7vbqZLe0toXnnkf+BETe8leYW37WPwkJimHjmw2YR/8AU3P/AMRXThqmIoVPaUN16hPkmvfPkDQ/EljrEklrbed58CfP5if3PkrarjPAej3un6rqU9zavbwTJJskk/j3z+ZXZ1+84Wc507zPnAooortEFFFFABRRUlvbvPJsT79LYAt7d55Niffqe4uEgj+y23+r/jk/v0XE6QJ9ltv9X/HJ/fqpWX8TVgFFFFbAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABU9vbvcP/zzRPvySfwUW9u9w/8AzzRPvySfwVJcXCeX5MP7uD/0OsW3sgC4uE8vyYf3cH/odFvcJPH9luf9X/BJ/cqpRR7NASXFu9vJsf79R1at7hJ0+zXH3P4JP7lQ3Fu9vJsf79NN7MCOiiitQCiiigAooooAKKKKACiiigAooooAKKKtQQJs86X93An/AI/Ut2AW3gTZ50v7uBP/AB+o7i4eeT/nmn8Ef9yi4uHnk/55p/BH/cqCs0nuwCiiitgCiiigAooooAKKKKACiiigAooooA9c+AP7S2veLPEmu23xE1HQNA0yGzjksnuIf7O3z7/3nzySfP8AJXuf/C2/Af8A0Pfhb/wdWv8A8XXwv4k8Np4kjgjed7fyX8z7nmVi/wDCtIP+ghJ/34r8yxXC7lW/2d2R6MMV3PtD4tfHfRPDvw51++8HeK/Dep+KIYEks7SO8hvnkfen/LFH3v8AJvr5m1X9qL4neKdPudF1K10r+zdSgksbnyNGkR/Ik/dybH8z5PkeuP0vwGml6laXqXzySQv5nl+RXV4r0cv4Zo043xGrM54qeyKPhj4jeJ/gIbq28EpDcJq2yS6k1Kykvvufu49nlyR7Pv17X8Af2lte8XeJNctviJqmgeH9Mhs45LKSeH+zt8+/5/nkk+f5K8iHSsbxJ4bTxJHBHJO9v5L+Z9zfXZmGQUMQpzp/GKnXnTPub/hb3gL/AKHvwr/4OrX/AOLr5Z+Nnxy8S+MvF3ibwLaz6Zf/AA/vZo7WG/sLTe7w+XG/yXSSbP8AWf7FeTf8Kzg/6CEn/fiul0fS/wCx9JgsUfzI4f8Alp/q68rL+GvY1vaYjU1niuc1/hr8XPE/wj1rSvC2ifY08K3ur28+o3epWck7pHNIiTyef5iRxoiJ/wAAr7Fb4ueANz7PHXhfZ/2GrX/4uviq8t/tljPa7/L85JI/Mrk/+FZwf9BCT/vxWmY8ORr1PaYd2FRxXJufoEvxc8Br/wAz14XP/catf/i6+UfFX7WnxMsfFniGz0hNFu9Gt9RuobKdNGkn86GOd4433+Z8/wAnl/PXlf8AwrOD/oISf9+K6zT7P+z9NtLXf5nkokfmVnguF4wnfEu4TxUzPk8Va1ceJP8AhaLRp/wm3nfbvL+yv9k8/wAvyP8AUf6z/V/9NK29W/ai+KPirT7jQtTtdKOm6jBJY3PkaNJG/kSfJJsfzPk+R+9R0ma+iqZJg6nJeHwHP7eoc5/wrvR/L2eZe/8Af/8A+111FnZvPJsT93Gn35JP4KfBbvcPn/Von35JP4KfPcJ5fkxfu4P/AEOvXjRhT0pmATzp5fkxfu4P/Q6q0UV1pWEFFFFMAq1BOk6fZrj7n8En9yqtFS1zASXFu9vJsf79R1at7hJ0+zXH3P4JP7lQ3Fu9vJsf79Qm9mBHRRRWoBRRRQAUUUUAFFFFAHV/CX/kqfhr/r6k/wDRD18cf8Fdf+TlvD3/AGKNl/6UXVfY/wAJf+SpeGv+vqT/ANEPXyb/AMFWtOtpv2lNFu75/Ls4vClllE+9K32i6+RK/AuNP+R7S/69f+3M+ny//c5/4jiv2L7NP+ED8aXNzvSzF9axg/8APR9knyV+lf7WmmajrfhfQI9O0+81CRNWaR47C1eeRE8iT+5X5qfsl6g+seF/GG5UhjN7aIkKfchTy5/uV9+v+2NqId3TwdZpj11R/wD43X895jWo08diqdT7fL/6Sfq+Bo16mDw1SjDn5OYxf2ZvDmr6X8VvtV9oup6fB/Zl0nn3djPBHvzH/HIle8fGq3nvPg/4wtbaCe8uJtP2Jb26eZI/zp9xKz/G/wAXLnwr8JNJ8aDTYbqS+Syk+wz3Lxxp5yb/AL/+xXE/Dv8Aaiu/HfjzRvD8nhyzsBqE3l/aoNReR0/dySfc8v8A6Z1lTVGhT+q8/wAZnUdbGVPrXJ8H/tp4l4R8I+Ibfxn4ceXw5rVvBHq9jI8kmlzxxonnx/7Fet/taAjxZ4FH+xff+j7eur+Ln7QF/wDDDxdHo0Ggw6sj2UN39onvXg+/JINmzy/+mdeG/Ef4tz/FvXPD89zpEOkf2Z5ifuLp59+943/55/8ATOvOr+wwmHlQhP3z2MP7fGYmlXqQ9w/ImT/WP9a/QDSf+SSeE/8AsGad/wCkqV8ASf61/rX3/pP/ACSTwn/2DNO/9JUr9ij/AMjbL/8Ar6vyPz7Bf7rjf8JveB/GGox6lpOiJ5P2DfJH9z95/wAtJP8AWV6xXh3gv/kcNF/67/8AtOSvca/qrD/AfmUwooorrMwooooAKKKKACiiigAq3b26W6edcfc/gj/v0QW6W8fnXH3P4I/79QXFw88m9/v1j/E0QBcXD3Em9/v1HRRWqVgCiiimBPaXb2b/APPSN/vxyfx1JcWibPOh/eQP/wCOVUqe3nezf/noj/fjk/jrFp3ugIKKtTwJ5fnRfPB/6BVWtU7gFFFFMAooooAKKKKACiiigAqS3t3uJNiffot7d7iTYn36muLhIE+zW/3P45P79ZNvZAE86QJ9mt/ufxyf36q0UVaVgCiiiqAKKKKACiiigAooooAjk1O90n/iYaVD5+pWw3wQeTJPvk/3E+/Th8XviOrf8gIfjodz/wDF1qeF/wB34q0nZ/z3/wDacley311NY6XfXgjM5trZ5gP9xN9fx94wT/4WqKX/AD6/9ukfvHh/i6NHAVYVKMJ+/wD+2xPBj8XviM/P9gr+Gh3P/wAXS/8AC4viKOP7BT/wR3P/AMXVmH9pzUWt0f8A4R21+dP+f9//AI3XZ/Cr4sXfxH1TULWXTI7H7LbJNmG5ebdvcp/Svwpwmldx/r7j9pr03hqUq1XAQ5Y/3/8AgnBN8X/iOzf8gJR9NDuf/i6G+L3xI286EB/3Arn/AOLr6A5Q9WP4VkeMNfl8M+FtT1VITdtY2vn+XI7oj45+/WcanM7JHjxzHDSlGMcHD8TxT/hb3xG/6Ai/+CS5/wDi6P8Ahb3xG/6Ai/8Agkuf/i60f+GmNQ/6Fm1/8GD/APxFPg/aS1OfeU8M2qRom95Hv32J/wCOVt7Op/K/u/4B7zwtZK/1GH/gcf8AMqWfxS+JV1OE/sWONP45JNGufl/8fpLz4wfETfst9CbyP9vQ7n5//H6+grh3EYUCRB33Y+aq6LIzFVDPWHtLM+c/tLDP3nhYfifPx+MHxGb/AJgi/hodz/8AF0n/AAt/4jL/AMwRf/BHc/8AxddT4z+O154Q8UajpC+H4Lr7MU/fPesknzoj/c2f7dZej/tIahqmqafZnw3Cn2i5hg3rfu+ze+z+5W/s52vy/wBfce9TozqUfrEMDDk/xr/MzP8Ahb3xH/6AQ/8ABHc//F0f8Le+I/8A0Ah/4I7n/wCLr6BkheNgrAp/vUxVcjKgv/u1y+11seL/AGlhP+gWH4nhdp8Y/iBdkQ3GjLH/AHHGh3Pyf+P1Xm+LHxJt5nR9DG9P+oNc/wDxdbOv/tDX/h/xBq2ljw5DP9huXtt76g6b9j/7lQ2f7T11dfubjw5axx/wOdQf5f8Axyuz2dSf2WexTwtVw9pDAw5P8cf8zK/4W/8AEf8A6AS/+CO5/wDi6P8AhcHxHH/MCX/wR3P/AMXXq/w/8XXnjLR576609dPkW5e2EcM7TJ8iJ8+//gddVz6t+VYS912aPLnj8LTlyywcPxPn3/hcHxGb/mBL+Gh3P/xdOX4wfEZf+YCv46Hc/wDxden/ABO+IF38O9Hsb2LThfm5ufs22ad4f4Hf/wBkrzxf2mNSbr4dtv8AwPf/AOIq+Sclfl/r7j0sMnjKXtqOAhy/4/8AglI/F74kNyNCH4aHc/8AxdI3xf8AiP0/sNf/AAR3P/xdex/D7xLN4y8I2esvbNaG4L/u0d3RNjun3/8AgFbcbbnb735VDqcvus82WZYalOUJYOHu+bPAv+FxfEZv+YCv4aHc/wDxdH/C3viMv/MBX/wR3P8A8XXoXxQ+KF38ObjTYotMF+b5JnzNdPDs2bP9j/briV/aY1Juvh22/G+f/wCIrRQm1dR0/ryPVw8XiqXt6OBhy/4/+CU/+FufEdf+YEB9dDuf/i6B8XviOemhA/8AcDuf/i6634e/GW88d+Jo9IfR47UPDNP5kNw833D/ALlepSSPGwUsyD2qJXi9V/X3HDicVSwk/Y1cHDm9f+CeAf8AC3fiO3TQh+Gh3P8A8XVuz+KXxFmjkmuNECQx/wDLOPRrnzH/ANz5692lZ7fTLq9wzpbwu+P7+xN9eDR/tPajcbZB4btuUz5f25//AI3WlOMpLRfgVhaix7ksNg4e75/5sik+MXxKZudCAT+BDodz/wDF0z/hb/xHb/mBr+Gh3P8A8XXc/C/4sXfxG1e/s5tJjsRb2/neZDctNv8An2f3K9Hzt7sPwrOU3F2kjGtjKGEn7Grg4c3r/wAE+f8A/hb3xH/6AQ/8Edz/APF0f8Ld+JB/5gQ/8Edz/wDF17N4w1ubwv4Y1PV0hN0bG28/y3dkR+P79eSr+0tqEf8AzLlsf+39/wD43VKMpK6X4f8AAOrByjjIuWHwUJW8/wDglT/hb3xIHH9gj/wRXP8A8XSf8Lc+JCnP9hZ+uh3P/wAXVi6/ab1BUeT/AIRy2AjTP/H8/wDPy695ZJPLVmDIKLSg05IzxlSOBUVXwcI83n/wT50/4SRNQaS61K6srfUpv3l1B5iQbJP+Wkex/uVYtbhLiMPDOlwn9+N99eOfEz/kp3jD/sMXX/oyu/8Ahn/yJdh/10n/APR8lf6JZDV58voR/uQ/9JP47x3v4qpNfzyOoooor6Q849V+GX/Imx/9fU//AKMrnfjF/wAhLTf+vGf/ANGV0Xwy/wCRNj/6+p//AEZXO/GL/kJab/14z/8Aoyv4d4V/5OhW/wCvtf8A9uPUn/APj23/AOPVP9yvoTR/+QLpv/XrB/6Lr57t/wDj1T/cr6E0f/kC6Z/16wf+i6/s7D/GefMt0UUV6hmVNU0uDWLX7Lc7/L3pJ+7fy6xf+Ff6P/fvf+//AP8Aa66QircFskEfnXH3P4E/v1yVaNKfvzGP8CfFDxH8EXt9H0FLb+w9S1CC6m+3WXnvvkkjgkk3+ZHsSNI6+yJ/ip4JWV3/AOEx8O7Ff/oLWv8A8XXxVqEj38c8byf66Py64GT4P2N5BJa/bfL3x7PM+ypXxOY8NuvP2tLQ76OJUFZn3v8A8KT0GT94l3fvv/efu7lP/iK5L4zfFiH4U/DV7fwZrWjah4j0u6tbJbGeZb2dFEoSbfDHJv3p/wCOV4zZft463p1rDap4N0OSO2RIPM/teb5tnyf88/8AYrzu30+PXvFl943bZb3etXU+pPZxp+7R55N/l+Z/H9+vBweU43G11DFfAjlo4bBYL38NDkPrv4LfHCx8U/DXSdQ8YeINB0vxJM919qtZLmGy2RpPJHH+5kfenyBK6S78O+GPiTcnWLTV01NEj+wvJpV4k8HyfPs+Tf8AP89fCWufD+11zVJ72af95N5f/LBJP+Wddd8PPj1e/s26Rd+GNM0PT9bgvbr+1ZLi7untXSR40j2eWkf/AEwrTFcP4jCOVbCs3n7DGQ9jioe4fWsnwL8PyK6G41Exv/08J/8AEV2Wqa5o/haxgfVdUstHgf8AcQSX10kG/wCT/br5B/4b+1//AKEnQ/8Awbzf/G6534nfF66/aS0nTtMvNJ0/SLHTbr+0UuLS6e6k37DB5bo8fyf6z79eZ/Z+Y4ucIV7/ADDC4bL8Dz/VYch9HfHT47QeEfh5d6l4J13QNY8Qpc2qJaxzJffu3k/efuY33/cqb4LfG+x8UfDXSdR8Z+INB0vxJM919qtJLmGy2RpPJHH+5d96fIiV8deH/Adr4f1WO+hn8yREkj/1CR/fqPXPh/a65qs97NP+8m8v/lgkn/LOvoP9V37Hlv750fWvfPef2iP2hNc03xdFoHhG80fWPC99pCJezwQm+dJJ5J0dPOjk+T5PLrwqP4b6PHB5CSXuxE8v/X//AGutLw34fTw3Yz2sM3mRvP5n+r2Vq7a+qy3J6OCo8k9WcdSv7QbHH5caJ/cp9FFfRnMFFFFMAooqS3t3nk2J9+lsAW9u88mxPv1PcXCQR/Zbb/V/xyf36LidIE+y23+r/jk/v1UrL+JqwCiiitgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAqe3t3uH/wCeaJ9+ST+Ci3t3uH/55on35JP4KkuLhPL8mH93B/6HWLb2QCTzp5fkxfJB/wCh1VoorVLlAKKKKYBVqCdJ0+zXH3P4JP7lVaKlrmAkuLd7eTY/36jq1b3CTp9muPufwSf3KhuLd7eTY/36hN7MCOiiitQCiiigAooooAKKKKACiirUECbPOl/dwJ/4/Ut2AW3gTZ50v7uBP/H6juLh55P+eafwR/3KLi4eeT/nmn8Ef9yoKzSe7AKKKK2AKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKnt7d7h/+eaJ9+ST+Ci3t3uH/wCeaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MX7uD/wBDqrRRWqXKAUUUUwCiiigAooooAKtQTpOn2a4+5/BJ/cqrRUtcwElxbvbybH+/UdWre4SdPs1x9z+CT+5UNxbvbybH+/UJvZgR0UUVqAUUUUAFFFWoIE2edL+7gT/x+k3YDqvg/Z/8XJ8NTv8Au4/tUn/A/wBxJXxl/wAFdm8z9pTw6e3/AAiNl/6Puq+ufBXiSDQfHGi6tf747CyeSR44E8zYnlyJ9z/tpXz7+3d8I9f/AGlPjRpHiTwatqdJg0K106R9SnS0kM6Tz8bH/wB9Pzr8U4py3G4rNYV6MOaHJ/7cz38HWo4fDTdSZ8tfAP43aZ8K9J1zTtQ0y7v31CaCaOS1lSPy9iSDncD/AM9K+xbiPy5JE/uV8u/8MD/FVm3bdDB9tWjrrx+zJ+0i/H/CYzf+FU3/AMXX5Hm3BWKzCv7ajHkn9rzPbyrxHyfAUXQrYmEl/iifffhj9qnSfDfhnRtLbwxqtw2n2UFq86XECRvsRE9f9itX/hsTSn+94U1of9vcH+Nfnb/wzL+0f/0ODf8AhUt/8co/4Zl/aP8A+hwb/wAKlv8A45XP/qjnXf8A8kH/AK+cKf8AP6H/AIGfVXxc+IVv8TvF6a1a2FzpypZQ2vkXTI7/ACSSfP8AJ/10rxD4l/GbTvg3faPJqel3eoi/33CfZplTZsk/2wa4T/hmX9o1enjCT8fFbf8AxdZWv/sb/G7xUYf7Z1ez1jyRiH7d4g87aP8AY3msqHAeNlX9rio88fQ6p+JmR06HssLXhCX+OJ8rzZaQscV9/aT/AMkk8J/9gzTv/SVK8RX9gP4pYyV0Pj/qKR/417bt/sfwjpfh25GNS0q2tbK6Ef7yPzIE2SeXJ/vpX3NPKMc8ywdVUZckJ+8eZlWf5Zi8PjIUMTCc+TuTeC/+Rw0X/rv/AO05K9xrw7wX/wAjhov/AF3/APacle41/SGG+A+RmFFFFdhmFFFFABRRRQAVbt7dLdPOuPufwR/36ILdLePzrj7n8Ef9+oLi4eeTe/36x/iaIAuLh7iTe/36joorVKwBRRRTAKKKKACiiigCe3nezk+T94n8cf8AfqS4t08vzrf95B/6BVSpILh7STen/A4/79YtO90BHRVqeBNvnW/+o/55/wByqtap3AKKKKYBRRRQAVJb273EmxPv0W9u9xJsT79TXFwkCfZrf7n8cn9+sm3sgCedIE+zW/3P45P79VaKKtKwBRRRVAFFFFABRRRQAUUUUAFFFFAGn4X/AORq0n/rv/7Tkrkf2jFUeNEz/wBApP4/+m01db4X/wCRn0n/AK7f+05K9K8ReE7DxBb3huNNsbu+e1khhnurZHdfkfZ8/wDvvX8jeLFdYfiDDVH/ACx/9KkfuPh/X+rYWtP+tY2PhSPZ5dafh/Z9qn/3K9ct/wBm/wATpBEj22i8p/z+f/YV3Hwp+C83hvVNWl8R6Zotzby28ccP3Lkq+/8A20qOIPEbAZrldXBQoS970/mPbyfhVZDj6WaPHRqez+xH7j0P4cn/AIt74Y/7Btv/AOiUr5v+Puz/AITzxOc/wRcb/wDp2Svqu1tYbG3jgiRYII02IiJsRErl/HXw7s/Fvh/XEt9N06TWLy18tLy6hTej/wC/s31+H5Xi1gcfSxL+zKMj6yOJiliIf8/YSh6c3X5HxbZ6f/aE+xNkccf7x5JP9WkdSapeQfZZLWz/AHdon/fyaT/npJXt158AfEoj+zW1loqWkfPF59//AG3+Ss+f9nLxRLC6JaaKH2d7z/7Cv6O/4itgJXj9Xl/4Ej86j4eyaUv7Rp/ib/7Ne0eJvEGzobGH/wBHGu6+PTlvhdfAjH+mW3/oyuy0vw/pehyPJp2mWNhJImx3tbdId/8A3xVrUNNtdVs/st5aw3cEnz+RdIjp/wB8PX8vzqqU2z9Or5hGpmEMZybcv/kp8E6ps/tK4z/0z/8ARdV7fZ9qg/34/wD0ZX0X45+A+p694w1O+0nTtFg06XZ5Cbkh2fu0R/k2f36zNH/Z88Q2up6feXFporWsN1DJNH9o8wuiP8/8Ff0PhPEnA4fK6WXzoSvGEY9P5eU+GxHB8cZj55p9ejDmlz8n+KXNynSfszsq2vijZ0+0W3/oDVf/AGkCf+EV0X0/tH/2i9enaXoenaItwNOsLOwSZ97x2sKQ7/8AvijU9IsdbhihvrO3v40fekd1CkyI/wDwOv54dZe2v0PuamZRnmH17k/rlsfBVxs+1T/79V/kr6C8Sfs/65qHibVrqwsNFhsrm7ee2US7Aib/AJPk2fJWf/wzl4o37fsWi/X7Zx/6BX9L4LxRwGHw0KH1eXuQj9pH53ieBljK08T9eprn962v3HEfDu4huL7SbW8k+5qkHkSf3P38f/jlfbDRGPUip/56f+z15V8LPhXF4a0Foda0vSJr/wC2PMk0cCTbE+TZ8+z/AGK9UjnDZSV/9yT+5X4Rn+Z083zGtjKcOTnkfWyX1fC0cFGfP7GPLfv5nwV4g2eWP+vp/wCP/rpWF8lfXvxW+EcfiHSrG30DRdItb2G98+Z/JSHemx/40T++9ean9nHxP/z56Kf+3z/7Cv1bhTj7CZDl0MHUoSn9x5ufcPR4nxksxWLjQ5vsSOW+B5X/AIWB4SH8P2887/8Apm9fW2pqI9D1ED/n0m/9AeuQ+Gfw/tPCXhvS4L/TNLGt2fmO13BCjv8AffZ8+zf9x67jyE8iSJz5qP8AJtr8eznGQzDH1cZT+1KUv/ApHu1akFTw+G/59QjC/wDNy/aPgnVNnk2P/XH+/wD9c6zPkr6q+KnwbbxM+i/8I3pOjWqQJOkwKLbb9/l7PuJ8/wBx64b/AIZz8T7kH2LR/m/6e+n/AI5X7Twz4h4PJcqo4CpQlPk5u380pHzeecLx4kzCrmv1uNPn+xJ/yx5f0PHNI2/2gmOvl19jfAuKFvhdoUk2BGPP4/v/AL564f4Z/Ae98O+KItS8Qabos2lJbTRmNGSbe7n5PkdK9ot7G2tIYorO1htLWP8A1cMCbET/AIBX57xjxFQ4hx0MZTp+z93l/wDSj1MFho5Plf8AZXtPae9z88f8PLynzv8AtHSfaPH0Ibp/ZUf7v/ttNXgkezZH/uV93a74a0vXPtjXWk2F9dtavCk93bI7p8j7Pn/33r59g/Zx8URQqj2uil407Xn/ANhXvcFcZYfhmjWhXp8/Py/qc2cZPHiaGHpxrxoexj9r7fMeSaLt+1SY6+XX2r8PWR/h/wCFw/Uabbdf+uKV538K/gpP4Z1jUZfEel6Lc2c1tsij+S52Pv8A7jp/cr2KCztbeGOCKOOCGNdiRxpsjRK+U4uzuln2ZTxlCHJ8J2YXDxyrLoZVz+05JSlzx68x8r/tBNG3j3xJ6+Vb/wAf/TsleRfJX21478A2Xizw/rkcGm6a+sXlr5cNzPCm/f8A7+zfXi6/s6+J5FJFlowx/eu8f+yV9pwZxthOHMHPD16Ep88uY4s7yNcTSo1Fio0PYxjD3uvmeGybNklfQ/7N+3/hLNc2/wDQNT+P/pslY0/7OPiiSCVEstFDhMf8fn/2FfRWk6BpeitKbHTLCxkZAjyWtskO/wD74rh424ww/FCw3sKfJyc3/k3L/kaZRlMeGcLiaDrxr+25fh+xy/53Pi/4mf8AJTvGH/YYuv8A0ZXoXwz/AORLsP8ArpP/AOj5K89+Jn/JTvGH/YYuv/RlehfDP/kSrD/rpP8A+j5K/r7h7/caH+CH/pJ/NuL/AI9T/GdLRRRX1Zxnqvwy/wCRNj/6+p//AEZXO/GL/kJab/14z/8Aoyui+GX/ACJsf/X1P/6MrnfjF/yEtN/68Z//AEZX8O8K/wDJ0K3/AF9r/wDtx6c/4B8e2/8Ax6p/uV9CaP8A8gXTP+vWD/0XXz3b/wDHqn+5X0Jo/wDyBdM/69YP/Rdf2dh/jPPmW6KKtQwJbp51x9z+CP8Av16bdjMWC3S3Tzrj7n8Ef9+oLi4e4k3v9+i4uHuJN7/fqOs0nuwCiiitQDYn9yiiigAo8uiigA2J/cp9pN9kfeiJ8/34/wC/TKKnlTAt3Funl+dbfvIP/QKqVJBcPaSb0/4HH/fqe4t02edb/wCo/wCef9ys03HRjKlFFFbCCiiigAooqS3t3nk2J9+lsAW9u88mxPv1PcXCQR/Zbb/V/wAcn9+i4nSBPstt/q/45P79VKy/iasAooorYAooooAKKKKACiiigAooooAKKKKACiiigAooooAKnt7d7h/+eaJ9+ST+Ci3t3uH/AOeaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MXyQf+h1VoorVLlAKKKKYBRRRQAUUUUAFWoJ0nT7Ncfc/gk/uVVoqZLmQD7i3e3k2ScSVGfarcdzHJD9nupPkT7k/9yuv8N+DLnS9Qn/ta2tbi3dPk/wCWlfFcRcTYPhnCfWcV8X2Y/wA3+E9jLctrZpV9lROFIHfrSfeBHSt3xL4YvLG4vb90hjsRP+78uT/2nWHx96vTyXOcHnuDjjcJPmT/APJZfynLjcHWwFb2NaAtFFFfSHCFFFWoIE2edL+7gT/x+k3YBbeBNnnS/u4E/wDH6juLh55P+eafwR/3KLi4e4k/55p/BH/cqCsknuwCiiitgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACp7e3e4f/nmiffkk/got7d7h/8Anmiffkk/gqS4uE8vyYf3cH/odYtvZAJPOnl+TF8kH/odVaKK1S5QCiiimAUUUUAFFFFABRRRQAUUUUAFWoJ0nT7Ncfc/gk/uVVoqWuYCS4t3t5Nj/fqOrVvcJOn2a4+5/BJ/cqG4t3t5Nj/fqE3swI6KKtQQJ5fnS/u4E/8AH6tvlAW3gTZ50v7uBP8Ax+o7i4eeT/nmn8Ef9yi4uHnk/wCeafwR/wByoKzSe7AK3NA0tL+3gRYfPupptif7b+Z8lYddl4C/5CWif9hOH/0elcmNqeypOS6HxvFcpxwMIwnyXnE9s+E3wTW9XUD4v0JowGQ2xe55/j3fcf3r0T/hQvgVRk6Lx73M3/xdegbTtG0gZ5ORWVrXiLSvD9usmqaha2CN8qNdSrHv9vmr8AxGaY7E1ueFSSb/AJWfW4PhjJsuwqjUowly/alGJ5t4m+F3wy8H6aL/AFayNnaM6xBxNcN83YYVs1haTpPwY13Ubaws4vPurhtsKb7pN9cL45+IHiLxzqGo6OsSahYwXztbR6fAZZCiyPsbKf7AFd78Kvhzo+j6DpPi7XWvNIv7Z5JGF5KYVUb3RPMR/wDYNfTVKdbCYNTxOJn7R7KMvzPz6jicNmmZujlmCo+wh8c5w/8AArHWX3wX8AaZZzXd1o6w29tG0zv9rm+VUGc/f9K4fzPgSV38Aev+mVV+LfxgvzrFzpei31heaJdWe15EjWX75dH+dH9hXi20mPy8fu/uV25ZleKxVH2mJrzXb3vzueRn/EGV4HFewy3BUZ8nx80Y/wDkvKe9/FL4JwW2m6cfB2gySTtN+/2XP/LPZ/tvX5weNkmtPiR4rs5Tse21e+gdP7kiTvX6W/BX4l654x8QX9lqUtvLbw23mr5ERT5txr83vih/yWn4i/8AYz6p/wCl09fScLyxKq4nB4z3+SDffqff5LhcsxCnm+Xw5Iz93k92MfdLXgfw/qkmuaLqKWT/AGDz/M+0b08v/lpHXsWa8k8F+NL6zvtN0t57W30nf88kieXsj/eSf6yvVbO8tdQj3208NxH9zzLd98dfWYbl5ND6KZPRRRXaZhRRRQAVbt7dLdPOuPufwR/36ILdLePzrj7n8Ef9+oLi4eeTe/36x/iaIAuLh7iTe/36joorVKwBRRRTAKKKKACiiigAooooAKKKKAJILh7STen/AAOP+/U89unl+db/AOo/jj/uVUqS3uHs5N6f/t1i073QEdFWp4UkT7Tb/wCo/jT+5VWtIy5gCpLe3e4k2J9+i3t3uJNiffqa4uEgT7Nb/c/jk/v1Db2QBPOkCfZrf7n8cn9+qtFFWlYAoooqgCiiigAooooAKKKKACiiigAooooArX95qOnW8t1pcsianD+8gdPL+/8A8DrPHjz4mKu1dVuR/wBsbX/4itdcnk8UvFfmXEnAuX8TYiGKxs5c0Y8vu8v/AMifd5DxbicgoToUaFKfPLm96PN/7ccvqnxe8eaHJAmoeIZ7PzvueZDa/P8A+OVQ/wCF/eK/+hrk/wDAaH/4is74sWdzeX2k+TazXEaQT7/s8Dyf8tI64/T/AA3qmoT7P7Ouo40/ePJJav5aV8TPwkyOH26v/kv/AMifWf8AES8f/wBAWH/8B/8Atj0iz+N/i+8f5PFbxxp+8eSS2h8tE/79066/aC8Tf6mHxRNHEn8bW0O9/wDb+5Xnl5b3vl/ZbPS9QjtE/wCnV/Mnk/56SVR/sfUf+gde/wDgK9Zw8J8m/nq/+S//ACIf8RLxl7/U8P8A+A//AGx6R/wv/wAWf9DXJ/4DQf8Axuj/AIX/AOLP+hrk/wDAaH/43Xm/9j6j/wBA69/8BXo/sfUf+gde/wDgK9H/ABCPJv56v/kv/wAiH/ETMZ/0BYf/AMB/+2PSP+F/eK/+hrk/8Bof/iKP+F/eK/8Aoa5P/AaH/wCIrzf+x9R/6B17/wCAr0f2PqP/AEDr3/wFej/iEeTfz1f/ACX/AORL/wCIl4//AKAsP/4D/wDbHpP/AAv/AMV/9DXJ/wCA0P8A8RSf8L+8V/8AQ1yf+A0P/wARXm/9j6j/ANA69/8AAV6P7H1H/oHXv/gK9H/EJMn/AJ6v/kv/AMiT/wARLx//AEBYf/wH/wC2PSP+F/eK/wDoa5P/AAGh/wDiKP8Ahf3iv/oa5P8AwGh/+Irzf+x9R/6B17/4CvR/Y+o/9A69/wDAV6P+IR5N/PV/8l/+RK/4iXj/APoCw/8A4D/9sek/8L/8V/8AQ1yf+A0P/wARR/wv/wAV/wDQ1yf+A0P/AMRXm39j6j/0Dr3/AMBXo/sfUf8AoHXv/gK9H/EI8m/nq/8Akv8A8iR/xEvH/wDQFh//AAH/AO2PSf8Ahf8A4r/6GuT/AMBof/iKP+F/+K/+hrk/8Bof/iK82/sfUf8AoHXv/gK9H9j6j/0Dr3/wFej/AIhHk389X/yX/wCRD/iJeP8A+gLD/wDgP/2x6ra/tA+JpE8mfxW8f9yc20Pyf+OfcqG8+OXjKzneCbxRPHJ/17Q//G68w/sfUf8AoHXv/gK9aVnZ3V5B9ivNOvY4/wDlhcfZX/cf/YUT8J8m/nq/+S//ACIoeJeM/wCgPD/+A/8A2x2v/C/fFf8A0Ncn/gNB/wDEUv8Awv7xX/0Nkn/gND/8RXnVx4f1Szmkgm0u98xP+nV6Z/Y+o/8AQOvf/AV6X/EJsm/nq/8Akv8A8iaf8RLx/wD0BYf/AMB/+2PSP+F/eK/+hrk/8BoP/iKu2vxu8VfZzdT+K5o7VOmbaH53/uJ+7rzjT/Dd7JHJdXmnXsdon/Tq/mP/ANM46gvLfVNQk3vpd7HHH+7SOO1fy0jqP+IT5N/PV/8AJf8A5Eh+JWPf/MFh/wDwH/7Y9Hn/AGgfFczbx4okRP4I0tofk/8AHKh/4X/4r/6GyT/wGg/+N15v/Y+o/wDQOvf/AAFej+x9R/6B17/4CvV/8Qmyb+er/wCS/wDyIf8AETMZ/wBAWH/8B/8Atj0j/hf3iv8A6GuT/wABoP8A4il/4X/4r/6GuT/wGh/+Irzb+x9R/wCgde/+Ar0f2PqP/QOvf/AV6f8AxCPJv56v/kv/AMiH/ES8f/0BYf8A8B/+2PSf+F/+K/8Aoa5P/AaH/wCIpP8Ahf3iv/oa5P8AwGg/+Irzf+x9R/6B17/4CvR/Y+o/9A69/wDAV6P+IR5N/PV/8l/+RH/xEvH/APQFh/8AwH/7Y9J/4X/4r/6GuT/wGh/+Io/4X/4r/wChrk/8Bof/AIivNv7H1H/oHXv/AICvR/Y+o/8AQOvf/AV6P+ISZP8Az1f/ACX/AORF/wARLx//AEBYf/wH/wC2PSf+F/8Aiv8A6GuT/wABof8A4ik/4X14q/6GuT/wGh/+Irzf+x9R/wCgde/+Ar0f2PqP/QOvf/AV6F4SZMteer/5L/8AIh/xErH2t9Sw/wD4D/8AbBrGoT6xrF/qNzP9onup5J3uP77vXrfw0/5Eqw/66T/+j5K8k/sfUf8AoHXv/gK9etfDuF7fwjYRzRyW7o8/7uRPLk/1klftuAoLDU6eHh9iJ+S4qp9YqTqfznTUUUV7RxHqvwy/5E2P/r6n/wDRlc78Yv8AkJab/wBeM/8A6Mrovhl/yJsf/X1P/wCjK534xf8AIS03/rxn/wDRlfw7wr/ydCt/19r/APtx6k/4B8e2/wDx6p/uV9CaP/yBdM/69YP/AEXXz3b/APHqn+5X0doFulvoemzXH3PssGxP7/7uv7NoO0zz5lyC3S3Tzrj7n8Ef9+oLi4e4k3v9+i4uHuJN7/fqOvQSe7MwooorUAooooAKKKKACiiigAooooAKfBcPaSb0/wCBx/36ZRSavowLdxbp5fnW3+o/jj/uVUqS3uHtH3p/+3U9xbpJH9qt/wDUfxx/3Kxv7PRjKlFFSQQPPJsT79bX6iC3t3nk2J9+p7i4SCP7Lbf6v+OT+/RcTpAn2W2/1f8AHJ/fqpWX8TVgFFFFbAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABU9vbvcP/AM80T78kn8FFvbvcP/zzRPvySfwVJcXCeX5MP7uD/wBDrFt7IBJ508vyYvkg/wDQ6q0UVqlygFFFFMAooooAKKKKACiiigAooooAa0RkjCt96us8N/EWGxmkh8T6stvaMnl2r3Cfx/8AAErk8Fl561zHjrP2Wxz/AM9//ab1+W8f4LDYrIMRUr04znD4f7vvR2PueC6c8RnmGwvPyQn/APIyPa5PAPj/AMZ2rXuk6Nc6x4euyk1nPFPapHLD1R03yI4qp/won4j7v+RPvdv/AF82v/x+vpz9mVc/ADwFnkf2Rb/+gCvTizHhcZHUV8NkvENXK8DDD4TDwhBf3f8AyZ67meb4W+OrRqTlLlnL/wBKPhcfAr4i448G3hH/AF9Wv/x+j/hRXxFz/wAibeZ/6+rX/wCP19peJryXSvD+q3tvt8+3tZZo9y8b1QmvA/2Qfjd4q+MsfiSTxJ9hMVnFaPbfY7ZoRlzNvzmR/wC4lexPjzFU60KEow5peUv/AJIMPw5UxWBrZhCXuUeXm/7ePMLP4D/ELzN8vhC8RI/+Wf2u1+f/AMj0T/BH4kXD5/4Q25jj/gT7Va/J/wCR6+6KWt/9ccdf4I/+Tf8AyR4/9n0z84td0HU/CurT6bq1m9hqMMaO8Dukmzf9z7lUf469J/aO+X42a4P+mNr/AOi683/ir9YwFaeLwlOtU3lFHkVIeznyC0UUV6hiFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFW7e3S3Tzrj7n8Ef9+iC3S3j864+5/BH/fqC4uHnk3v9+sf4miAjooorYAooooAKKKKACiiigAqe3t3uH/55on35JP4KLe3e4f8A55on35JP4KkuLhPL8mH93B/6HWLb2QCTzp5fkxfJB/6HVWiitUuUAooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFW7edJ0+y3P+r/AIJP7lVKt28CbPOm/dwJ/wCP1jU2AfHp/wBn8x7v93An/j/+5UFxcPNJ/wA80/gj/uVP/aCah+5uNkcf/LCSP+CqlxbvBJsf79TTvf39xjKKKK6BCNXaeAf+Qpof/YUg/wDR6VxbV2fgL/kJ6J/2E4P/AEeleZmP+7S9GfF8W/7rT/xwPuuSZYw5dlUKARk+1fIHj/4tXvxE0+yhvrG3sUtpPPMkLu/8BT+tdj+1A0bax4dmZFkAtLk8/wC/HXc/Cv4QXfw91q81G8vob9bi28oRxQmPb85fPU+tfj+Ajh8rw8cfW96cvgXpoyc6rZlxDjp5Nhv3dGHxz+L4vej7pnfBf4WQaXHpPi9dQmk+12CuLMqmxfNVHPzY9q5740/Fa8mvvEPg8WVubDCILrzG3/cR+1ZXxh+K8PjKzu/Dp0p4FstSdRcySo6OYXdPuV5Ysfk8pXu5fllbGVvr2O3+yvxTPiM9z/DZfhv7FyT4ftz/AJvsyjyyHUUVHJJ5ccj/ANyvudlqflSTk7Lc9f8A2aT/AMVlqxPawQ/+PGvgf4pf8lq+IX/Yz6p/6XT1+mvwt+Dt74B1q91G51KO8Wa28oJHCyYy5b++fWvzK+Kf/JaviH/2M+qf+l09eJw7iKWKzPGVaL09n/kf1PwzgcRl+S06GJhyT5pGDXrnwr/5FWT/AK+pP/ZK8jr1z4V/8irJ/wBfUn/slfSUfjPfmdfRRRXqGYVaht0t4/OuPufwJ/fpYLdLePzrj7n8Ef8Afp//ACEv+vr/ANDrBzvp0GVZ53nfe/36joorZeQgooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAPt7h7STen/7dW/7P+2fPafc/jj/ALn/ANhVS3t3nk2J9+rf2z+z/ktP+Bv/AH//ALCuepv7m4xlxOkCfZbb/V/xyf36qVbuLdJE+1W33P44/wC5VSqp7CCiiitgCiiigAooooAKKKKACiiigAooooAKKKKACiip4IXnf/nmiffkk/gpN21ALS3e4f5P3caffk/uVJcXfyeTDvjgT/x+i4uE8vyYf3cH/odVKys5asAooorYAooooAKKKKACiiigAooooAKKKKACiiigAooooAt29wlxB9muH+T+CT+5T47D7P5j3e+OBP8Ax/8A3KZb26eX5037uBP/AB+n/wBoJf8A7m42Rp/yxkj/AIK5Hf7AyCe6ed/+eafwJH/BUFSTwPA+x/v1HXTG1tBBRRRVAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAeq/DL/kTY/+vqf/ANGVzvxi/wCQlpv/AF4z/wDoyui+GX/Imx/9fU//AKMrI+K0aR32mTTfvP8AQp9lv/f/AHlfw5wt/wAnPrf9fa//ALcerL+CfIOl2aWemwXt5H+72fuIP+e3/wBhXvGn3D3mm2Ej/fe1g/8ARdeHW/8AxUFrH/yz1LZ/q/8AlnPXt2kf8gXTf+vWD/0Cv7LwnxnnTLdFFFewZhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABUlvcPaPvT/APbqOpIIHnfYn36TtbUC1/Z/2v57T7n8cf8Ac/8AsKZcXCW6fZbb/V/xyf36f9s+w/Jaf8Df+/8A/YUy4t0kj+023+r/AI4/7lckd/f2NSpRRRXYZBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVPb273D/880T78kn8FFvbvcP/AM80T78kn8FSXFwnl+TD+7g/9DrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAooooAKKKKACiiigAooooAT0rmPHf/HrY/8AXf8A9pvXT+lcx47/AOPWx/67/wDtN6/PuOv+Sexn+H/24+/4D/5KXB/4v/bZH33+zBlvgD4BPppFuP8AxwV458XP2w9Z+HvxZ1zwhHpmlzwWZhEM090yP88KP86Y/wBuvZP2Y8N+z/4CAOP+JPbZ/wC+BWz8WPAo+JHw/wBe0CKWOzu9QtXt47hk+5kda/n6rCvUwcPq0+SR9PRxOBw2e4j+0KPtITnJb25fe+L5dj5T1T9tTxVfabe2d/4X0iC0mheDzI76bzH3p/B+7ryv4O/HbVPgrNqMmlaTZ6gt7DbQeXczMkcUcQfZs2D/AG6T4teALn4X+KY/DWuX9pdSSWiXkV1YwOiQq7umz5if7lebah/xL1nMx4hTfhK/LcVj8xp4r3p+/A/pPKsiyLEYGcMPR/c1rfze9Y+mpP2/PFVu2+bwrocY/wCml9N/8br60+EfjSb4ifDfw94iureO2m1KxhunhhOUXegPHtXiP7NP7NOqfDHxc/iW/wBf0/UrO50v7OlvBbvG6u7o+fn7fJX1AF2qPUflX6Vk8cb7P2mMndn868W4jJFV+rZPRsl9u8tf7vLI/On9rT4jXXh79obxJp0On21wiW1kfMld+8Ga5HwX4kn8UabcXU1rDb7J/I8uN9//ACzj/wDjlP8A21v+Tm/Ff/XrY/8Aoisb4R/8i3d/9f0n/ouOv6kyWUvqlFf3In4zX+OZ29FFFfTHMFFFFABRRRQAUUUUAFFFFABRRRQAVaht0t4/OuPufwJ/fpYLdLePzrj7n8Ef9+n/APIS/wCvr/0OsHO+nQZVnned97/fqOiitl5CCiiimAUUUUAFFFFABU9vbvcP/wA80T78kn8FFvbvcP8A880T78kn8FSXFwnl+TD+7g/9DrFt7IBJ508vyYvkg/8AQ6q0UVqlygFFFFMAooooAKKKKACiiigAooooAKKKKACiiigAooooAntESTe8r/In/LP+N6J7h53/AOeafwR/3KgoqeXW4BVqCdJ0+zXH3P4JP7lVaKGuYCSeB4H2P9+o6nkneSFIW/ebPuSf3Kgoje2oCNXZ+Av+Qnon/YTg/wDR6VxjV2fgH/kKaH/2E4P/AEeledmP+7S9GfF8W/7rT/xwPtbVPC2i62sL6npVnf8Akjan2u3STA/4EK+fPid8aIPGml6cmgS6npbwzb3wfJ3x7D/cf6V1Hx+8Wa14Z1bQ4NM1GWyWaCZ5EiCHfhox3+p/OvAU/d81+YZDlKqxWKru6+z+p4HGnE06VWWWYKHI/tv/AMB5SSSR5pnmld3nZ97vI+93eozRVrT9PuNWvorKziE15cNtihMnl7hX6G3GlG72PxOnTqVqns4asjtbR9QvILaA4eaZIE8z++77K+hPhf8AB2TQbXUz4o0zS9USRle3GPtG0Y+fJdB3qj8N/hRZaL4fuNT8aaLHbXun3DXUU8jh3WJCro+UY/3B+VYnxY+NT311pz+DdeuI7JIZBdCODZ82U8v/AFif79fBY3F181q/UsF8HWX/ANsfr2U5Xg+GsL/ambfxPsQ+1/L8Mj1DwL8YtG8f3stnptrfW0kcXm/6VCqqU3lOPm9q/LL4pf8AJaviH/2M+qf+l09for8Dfh/4h8KeIL671PTGsLaW0CI7TI+W35/gY1+dXxS/5LV8Q/8AsZ9U/wDS6evc4VpYejjsXDDO8PZ/qj9ZyfFY7H5ZCtj4ck/8PKYNeufCv/kVZP8Ar6k/9kryOvXPhX/yKsn/AF9Sf+yV9ZR+M7JnX1bt7dLdPOuPufwR/wB+iC3S3j864+5/BH/fqC4uHnk3v9+u7+JojMLi4e4k3v8AfqOiitbWAvf8hL/r+/8AR3/2dUaKvf8AIS/6/v8A0d/9nWP8L0GUaKKK3EFFFFABRRRQAUUUUAFFFFABRRRQBPZ6fdag2y3ge42f886gkje2eRJkeORPvxyV2tvqH/CP6J4f+zbEgunj8+f/ANDrC8RxvqHiS++yJ9r+4/7j5/4K8qji3Oo19k19mY9FFFeqZElvcPbvvSp57dLhPOt/ufxx/wByqlSW9w9vJvT79ZNPdAR0USSeY7vs8v8A6Z0VqAUUUUAFFFFABRRRQAUUUUAFFFFABRRU8ELzv/zzRPvySfwUm7bgFvbvcP8A880T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVSskm9WMKKKK2EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABU9oiSb3lf5E/5Z/xvUFFJq4E89w87/8APNP4I/7lQUUUJW0QFq3uEnT7Ncfc/gk/uVDcW728mx/v1HU8lw8kKQv+82fck/uVlZp6AQUUUVsAUUUUAFFFFABRRRQAUUUUAFFFFABRRVqGBLdPOuPufwR/36TdhnqPwtj8vwTbvN/z9T7I/wC/+8rl/jLI8mqabI//AD5T/wDoyul+HNw9x4Rid/v/AGqf/wBGVzXxi/5CWm/9eM//AKMr+IeFZf8AG0K3/X2v/wC3HqS/gnx7Z/8AHpH/ALlfSWmf8TbR7D/n++ywf9tv3f8A6HXzbb/8eqf7lfQmj/8AIF0z/r1g/wDRdf2TQV5nnTLdFXv+Ql/1+/8Ao7/7OqNeipXMwooorUAooooAKKKKACiiigAooooATrViz0+51CTZbQPcbP8AnnUFdrb6h/YGieH/ALNsjguXj8+f/wBDrz8VXnRtyGkIHFSRvbvIkyPHIn345KRa2vEUb3/iS++yJ9r+4/7j5/4Kxe9dGHqe1ppkC1Jb3D2770qOit99xFu4t0uE863+5/HH/cqpUlvcPbyb0+/UcknmO77PL/6Z1nTvsAUUUVqAUUUUAFFFFABRRRQAUUUUAFFFFABU9vbvcP8A880T78kn8FFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6xbeyASedPL8mL5IP/Q6q0UVqlygFFFFMAooooAKKKKACiiigAooooAKKKKACiiigBPSuY8d/wDHrY/9d/8A2m9dP6VzHjv/AI9bH/rv/wC03r8+46/5J3Gf4f8A24+/4E/5KXB/4v8A22R9+/swtu/Z/wDAR9NHth/5DFJ+0tqV1p3wL8ZXdjcz2t1FpsjwzWrukiNjgq681wXwE+OHgbwz8GfBemaj4hht7630u2img8mT5HKf7lX/AB9+018Ede0vUvC3iDxdbS2l1B5N1bvBcruR+25E7+1fi1PK8fXwcfZ0Z+9H+V9jvxOLo4XPJ1qvwwq6/wDgR8ofDn4OeOfjYf7dtruLVbSwv47OeTWtSmeb5DHM8fzo/wAmx/8AyI9fR37Q37ML+OJPD8PgzRNA0uO2iuUum2/ZuXEewgInzfcenfD79oT9nL4X6TPpvhnxLZ6baXVz9pkiWK8ffLtRN3zp6Ig/CuqX9sz4L7izeO7Vj0H+iXI/9p15OH4TqwwzpVqM+efxaM+mzDjrEVcfTxODfJTp83Iuyl/NqeFfsda34iuPjFqWnatq+rXlvHo0u22vL6aeBXSeFN6b3NfdG07iMDFfKPgX4vfsy/DnVH1Tw94gs7G/aKSFpzHeP8jyb3T5x/fTNd+37anwVb/mfLYf9ulz/wDG69DLMjzDB4dUa9Ob+Uj57iPOcJm2PeKwy5IO2mh8Wftrf8nOeLP+vWx/9EVjfCP/AJFu7/6/pP8A0XHTP2mvGejfEL46eIPEPh7UI9T0e6tbVIbqNHTfsg2Sffp/wj/5Fu7/AOv6T/0XHX9E5PCdPD0ITX2D81r/ABzO3ooor6c5gooooAKKKKACiiigAooooAKt29ulunnXH3P4I/79EFulvH51x9z+CP8Av1BcXDzyb3+/WP8AE0QBcXD3Em9/v1HRRWtrAXv+Ql/1/f8Ao7/7OqNFXv8AkJf9f3/o7/7Osf4XoMo0UUVuIKKKKACp7e3e4f8A55on35JP4KLe3e4f/nmiffkk/gqS4uE8vyYf3cH/AKHWLb2QCTzp5fkxfJB/6HVWiitUuUAooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV13gaVLe80eR3REj1CCR3k/gTz0rkall8QTaHol/IkMNx9lgnn+ff8/yb64sVRdel7NdT5XiLB4nG4OEcLD31OJ9o+IPDHhL4xPBdQammoT6ehTGn3YO3fj7+z/c/SvmqLwh4hVUD+HdW8zuP7Nm/+Irx7wN+214t8Bx3X9keGPD8n2zZI/2ie6/grqB/wUd+JAYkeEvC2T/t3VfIYXh3Pssc6NCmp0+nNJf5kZhwzh+IKcMTjf3Nb7XKd3/wiOun/mXdW/8ABdN/8RXuPwZ+ENnHpuj+IdSt7y01qF5H8iclNnzOiZTt8hr5R/4eNfEfdu/4RHwtn/fuv8KX/h478ScY/wCET8L/APfy6/wp47JOI8ZS9jGnGH/b6/zNMn4NyvKcT9ac+f1/9KPqn4zeKPGVrr11pOladcXej3Fois0OnSTcvvV/nT2wa5P4U/B+Dxdb6kniGz1LTVhKRRb43ty6FMN99fn5rwP/AIeP/Env4S8LH6vdUh/4KOfEg9PCXhYfR7n/AArmp8M8QUML7CjQjC/2uZf5npYjhrC47MPr2Mrc8F9iXwn3v4q+IHhj4e2dtN4n8Q6T4eimfyoZNUvY7VJX9E3uPyr8jPiBqFtqnxb8dX1rcxXVldeINUnhmhcSJMkl08iOjj+Cur+OX7Svif8AaC0vR7LxBo2k6Ymk3T3cMmmvM8jyPG8ez5/9+vN7e38v5697h/h+XDuFqYjGVLVp+5b9T7CtWU1yonr2r4T26W/g/wC03KfI91Jsj/v/AOr/APHK8ns7NLOD7beJ5kf/ACwt/wDnv/8AYV6j8N7x7zw5JPM/mSfapP8A0XHXbR/eTPOOsnuHuH3v9+o6KK9hKxzhRRRTAKKKKAL3/IS/6/v/AEd/9nVGir3/ACEv+v7/ANHf/Z1h/C9BlGiiitxBRRRQAVPp9hNqU3kW6eY/8f8AzzSoK6DR44P7FtfO/wCPSa98u9/9k3/7FcmIqOlC6NKZmvoc8ccjw3VnqGz7/wBkn3yJVGurSS9kmjS/SOOf9+6eXAifZdn3HR/7lc38l/qH7lPs8dzN8kf9zfXNha85/GOcB+j6ZJreoR2sT+X/ABvJ/cSt2TwhZXHnw6dqcc93D9+CSr76xpfhHUI7KGyfzNnzz/8ALSsLWNHfRJ01PTpvMsX+dJ4/4K5fb1Ks7p8iNOQNL1/+z4ZNM1Sy+0Wu/wD1f8aVbk8T6fpdi8GhWT27v9+eSqetatba1DBN9mePUf8AVzSR/cdKxutdNPCwqfvKisZ84tFFFeqZBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRU8ELzv/zzRPvySfwUm7bgFvbvcP8A880T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVSskm9WMKKKK2EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRVqGBLdPOuPufwR/wB+k3YBYLdLdPOuPufwR/36guJ3uH3v9+i4uHuJN7/fqOs0ne7Geq/DL/kTY/8Ar6n/APRlc78Yv+Qlpv8A14z/APoyui+GX/Imx/8AX1P/AOjK534xf8hLTf8Arxn/APRlfxFwr/ydCt/19r/+3Hpz/gHx7b/8eqf7lfQmj/8AIF0z/r1g/wDRdfPdv/x6p/uV9CaP/wAgXTP+vWD/ANF1/Z2H+M8+Zbq9/wAhL/r9/wDR3/2dUaK75RuZhRV7/kJf9fv/AKO/+zqjRGVwCiiitQCiiigCfT9Pm1ObyLZPMk/j/wCeaVPJok0aSPDc2eobPv8A2SbfIlaWjxwf2La+d/x6TXuy9/8AZN/+xVuOS9kmjS/RI5/37p5cCJ9l2f6t0f8AuV4dTFVPaWR0chyfSrej6W+ualHawv5f/LR5P7iUxdmoah+5T7OlzP8AJH/c3118msaZ4P1JLKKyfzNnzz/8tK661ecFyQXvkQgUZPCFlcefBpuppcXcP34JKoab4gOn28mmapZfaLXf/q/40o1fR5NDmj1TTpvMsZPnSeP+CjW9Xttchgn+yvHqP+rmkj/1bpXHG9VWn78H+BoW5PE2n6ZYvBoVk9u7/fnkrmjS0V6VGhChsZc4UUUV1mYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVPb273D/880T78kn8FFvbvcP/AM80T78kn8FSXFwnl+TD+7g/9DrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAooooAKKKKAN/wCH+maHrPi6C08QXSWmlNBO73El19k+f/ln89eq/wDCu/hBtz/wk1t9P+Eli/8Ai6+dfFEfmaS++uI+zwf3I/8Aviv5h8ROJsfkucQoYacoR5I/a/vSP1LhvIcPmWC9tVf2j7Pg+DXw/wBT0a+vtHubjU44Y5D59rq/nJvRN/l/JXzxYtutIHf78iIa9l/ZWjSP4L+JyieWPt97/wCksNeMaf8A8eMH+4lfc+HObYzNsPWqYqfP8P6ny/EODo4HEexok9FFFfsp8qbHgrRrbxB420HTLvf9lvbryJvLfy5PL8t69p1L4R/DDRZ0g1PVm0yd03ol9riQyOn/AD0+evIvhcc/E7wl/wBhFP8A0W9M/a6jR/ifpW9Ek/4kaf8Ao6av548Rs7x2TYiH1WtKHu/zf3mfc8N5bRzJyp1T1P8A4V38H84/4Sa2/wDCli/+LqhrXwf+D2sWqJL4og2o/mfufE0Jr5N+zwf3I/8Avij7PB/cj/74r8PxHGma4qjOjWnOcJ/3j9Ow3DNLB1Y4jDVOScf67n2L4d+EHwtu1t9K03Wjf3McfyQQa+k0+xP9hK+SP2ivDsHhz45eIdB0lJpLeBLEQRyO88nz2iPXpH7LMaJ8btM2Ig/0C9/9AFYHx/GP2vNSH/TTSv8A0kjr9g8Pc6xmZYmcK05ckIfB/wCAn57xPgKWBqc0Xec//tjxz/hH9X/6A+of+Ar0f8I/q/8A0B9Q/wDAV695or+hvqy3Pz7nPBv+Eb1f/oEah/4CyV9V/Dz9lj4dat8J/DHijxVda1pF3qGnQXF68+r/AGWBJn/30+SuM7mvYvi7mT9kvS9//PnpH/oyOvzDj7HYnK8BCvhZ8n/7J9HkOGhjsT7OZgf8My/Ab/obZv8AwrYa6Dw/8Gvgt4bsZLWz8UpJG7+Z/pHimGSvlD7PB/cj/wC+KPs8H9yP/viv5w/18zj/AJ/T/wDAj9V/1Swvf8P+CfYP/Cv/AIPqu5/E1r5f/Yyxf/F1jfGD4X+HfBPhexvdIjuvPnvY4JJLi6knj8vy5JP/AGSvkzULdP7Nu/kT/UP/AAV9u/tFNu8AaMR/z/Q/+kslfdcHcUZrm2a0qNetLk5v5j5jPsjw+W4fnjE+eqKKK/rU/Kh9nHHdX1pA/wDq5rqCN/8AceTZX0Hrnwb+Gvhxo01PUJdM8wuE+3a15O/Z/c314BppH9q6bj/n+g/9Hx13/wC2tGkl94L3oknzah/O1r8J8Rs6xmTKjUws5Q+L/wBtPsuG8vo5jX9hWOu/4V38H+P+Kmtv/Cli/wDi6534ieDfh5ovhO8u/D2s295qsbwbIE1pLr5PM/efJ/uV8wfZ4P7kf/fFafheNI9Yg2In3JK/Jcg41zTGZthqFerLknKP2v7x9vmXC+Fw+DrV4P4IyO6q3b26W6edcfc/gj/v0QW6W8H2q5/1f8Ef9+oLi4ed3d3+ev7Jv7TRH4sFxcPcSb3+/UdFFbJWAKKKKYBRRRQAUUUUAFT29u9w/wDzzRPvySfwUW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAooooAKKKKACk3UNW14b8Lza5++f/R7H/np/f8A9ysKlSFKHPMZjRxvPJshR5JP+ecdbtn4I1a8++iW6f8ATd676w0m10eDZbwpH/7PUl5qlrp6b7iZLf8A66PXh1MwqVHamjr9icfH8N5/49QSP/th/wDZ0yT4d3Uf+pvYZP8Aromyt2TxzpMf/Ly8n/bF6nt/GOk3n3LxI/8Af+Ssvb4zf9B8kDgdQ8N6ppfzzWvmR/8APS3+es2vZkkEse9K5rX/AAfDqkck9tst7r/xx66qGYO9qxlOiefUUSRvbzyQzJ5ciffjor3U76o5wooopgFFFFABRRRQAUUUUAFFFFAE9vdS+Yn71v8Avuvmm3/eQR19H1x/jjwW+uWNhBpUFlZvDO8j/J5H7vy/9iOuOtS5tjSB5JRXYf8ACq9e/v6f/wB/5P8A43R/wqvXv7+n/wDf+T/43XB7Op/KaHH0V2H/AAqvXv7+n/8Af+T/AON0f8Kr17+/p/8A3/k/+N0/Z1P5QOPrSs7NLOD7beJ5kf8Aywt/+e//ANhXU6f8K9Uj8ya5fT5Nn3Lfz3+f/wAh1HefDPxDeTyTzT6fJJ/13f8A+N0vZVHuBx95ePeTyTzP+8r1X4VyeZ4Vk2f8/T/+06wfD/w31TT9csLq8/s+S0hfzHj3+Zv/AO2fl16Tb28FnHshgS3j+/5caeXXXRo2M5klFFFegZhRRRQAUUUUAFFFFAF7/kJf9fv/AKO/+zqjRUlxcPcPvf7/APHJ/frFJx22AjooorYAq3pmpzaZJJs2SQTffgk/1b1Urd0/whPqlrazW16kkc3+u/6Y1y1504Q/eGlMqXGtp5EkFnZQ2Ec3+vkg/wBY6VNq2gpaWqahp0zz2P8Az0/5aJVjxBJZDydG06FJHR/Lef8A26r6fqF54T1B7a4h8yB/vwf3/wDbSvOj8HPT37dzU0yYfG9j/wAs7fWYU/77rCt9TvdLtbvT/wCB98bxyfwU/WI7Kz1COfS7p9jpvTy/4KzfvCt6FG8dfgM5zFooor1DIKKKKACiiigAooooAKKKKACiiigAooooAKKKnghed/8Anmiffkk/gpN23ALe3e4f/nmiffkk/gqS4uE8vyYf3cH/AKHRcXCeX5MP7uD/ANDqpWSTerGFFFFbCCiiigAooooAKKKKACiiigAojjeeTZCjySf88462fDfhebW/3z/6PY/89P7/APuV6Fp+lWujw7LaFI//AGevIr42FH3KerOmEDgbPwRq959+NLdP+m71pR/Def8Aj1BI/wDth/8AZ12F5qlrp6b7mZLf/ro9ZMnjnSY/+Xl5P+2L1531rFVPgNOSBgyfDy6j/wBTewyf9dE2Vjah4c1TS/nmtfMT/npb/PXfW/jDSbz7l6kf/XT5K1o3Eke9KaxuIp/GLkgeM0V6D4g8Hw6nHJPbbLe6/wDIb159JG9vPJDMnlyJ9+OvZoYqGIOecPZhRRRXcZhRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVNaeTHveb95s+5H/fpN2QEsFulunnXH3P4I/79QXFw9xJvf79FxcPcSb3+/UdZpPdgFFFFageq/DL/AJE2P/r6n/8ARlc78Yv+Qlpv/XjP/wCjK6L4Zf8AImx/9fU//oyud+MX/IS03/rxn/8ARlfw7wr/AMnQrf8AX2v/AO3HqT/gHx7b/wDHqn+5X0Jo/wDyBdM/69YP/RdfPdv/AMeqf7lfQmj/APIF0z/r1g/9F1/Z2H+M8+Zbooor1DMKvf8AIS/6/f8A0d/9nVGispRuAUU+ed55N7/f/jk/v0ytF5gFFFFMC3pepzaZJJs2SQTffgk/1b1Pca2nkSQWllDp8c3+vkg/1jpVvT/CE+qWtpPbXqSRzf67/pjTteaz3QaNp0KSSI/lvP8A7deO6lGpU0R0e+Q6voEdnax6hpsz3Fj/AM9P+WiVpfufG9kclLfWYU/77rN0/UL3wfqL21zD5kD/AH4P7/8AtpVfV0srXUEn0q5cRum9PL/gqeWdRpfdIYy31K9021utP/gfejxyfwVRoor1KdNU9jnCiiithBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABU9vbvcP/zzRPvySfwUW9u9w/8AzzRPvySfwVJcXCeX5MP7uD/0OsW3sgEnnTy/Ji+SD/0OqtFFapcoBRRRTAKKKKACiiigAooooAKTdQ1bXhvwvNrn75/3Fj/z0/v/AO5WFSpClHnmNHJ61byXliYLeB7id/8AlnGnmSVR0/4Z69efO8KWcf8A08V6Z8TPB93efD640zw3Imn3z3UD+f57wfu0k/efvK8P1D4f+OdLj33niiG3/wCumvz1/KniFhsNmWce2qT5Pdj/AO3H6vw5mFbC4L2NCjz+8fW/wT8Ez6X8DfGmnzXK/v21Dnyf79iiV8e2/wAD547GDydRtZPkj/1kHl19a/shxXsfwP8AFWn3+oR6pf3Wr3saS/annT57WBI/3j1j2/wD8VxwRp5+hfIn/QRf/wCN16/CGYYbKaM4e2j9k8LOKdbFV+ecD54+H/gPVPDfiK7eayTyPsMn+kW/7z/lpHXc817n4X+A/ivS76Sa5fSfLeDy08u+eT/lp/1zp/iD9nfV9Ujkms5NMt7v/ru/lv8A+Q6/WsHxTgI+5WrxPmJ5fW+xA8u+F+f+FneEv+win/ot6T9rSCWX4l6XiOR/+JOmPLTt501egeBvgn4o8PeNNF1O8fSfIsrrz5447p5JP/RdfMX/AAUb+Hfjjxp8cPD914V11NLsofDMEEifb3tcyfbbr+BP+ukdfjPiHiMHnWJgqNaPJyfF/wBvM+44YdbASl+65yp9jn/54P8A98UfY5/+eD/98V8y2/wG+MlxIsSeMI/Nd/LH/E7l615/8O7P4gfEfXb3StJ8U30dzZ273Usl5qkqR7EdE6/8DFfkVPIadSEp08VFqHlsfoU87nTnCFTDS98/S/8AZdt5I/jVpm6N0P2C8/g/2BXn/wC1p/ycZ4t/646d/wCkKVyn7BPwy8f+Ev2k9F1DxPr8ep6cum6h+7GrPP8A8sCPufiK+g/jx+y/48+JHxe13xLon9hHTL5LVIPt+ovBJ8kEaSfJ5b/xpX6jwFVwWUYqc61eDhyS1/7eifBcTTrY+UP3PJ/Uj5Ms7eD7VB+4T/Xx/wAH/TSvo+4/1z/79ZNv+xX8UI5433+GPkeOT/kMP/8AGK9fk/Z78VySSP5+i/8AgdJ/8Yr9/o8R5TFf7zE+C/s/Ez+weYKTuPpXs3xVjkk/ZL0vYnmf6HpH/oyKsT/hnfxfz++0X/wOf/4xUv7WnhDXJv2Mbnw3pt7Daa/AukQedHdOiJIk8e/95X5nx/mWAzbBwoYevH+on0WQ0a+ExPPOB82fY5/+eD/98UfY5/8Ang//AHxXyf4k+E/xc8L6DqOs3Pi1pbWxh86b7PrErybPpWZ8PfBXxQ+JWiTaxpXi2eK1jufsr/a9XljfftV+P++/51/PSyGjye3+tR5D9X/tutz+w+rS5z671Czn/s27/cv/AKiT+CvpT9uqKN/hL4M3Jx/bsP8A6Q3FfmlcfA/4vQ2s7zeLY5I0Vi6f23Kd/wCFfq5+0p8K/EHxi+H3hjSfDr6f9usb+G+n+33XkJs+wyR/f8t/+Wj19dwjSwuV5lCvKvGcD5riCvWxeH5HRlA/Pr7PB/zwT/vivY/hhH5fg2DZ/wA95/8A0ZWt/wAMT/FD+/4Y/wDBw/8A8Yr0bwX+y/458P8AhyCyvJvD/nxvJJ+71GR/+Wn/AFwr+n6PEeUxn/vMT8r+pYn+Q5XTc/2rpuf+f6D/ANHx16J+2dHJJqHg7Yjyc6h/q/ra1PZ/s/8Aiu3vrSd59F8uG6gkfy7p/wCCTzP+eFeZf8FMvB/izxxJ8Mv+EU1UabJC+rvPJ9sa23bzalPuf8Dr8g8RsZgs5VGnRrx+1/7afZcNwr4Guqk4Hm32Of8A54P/AN8VX1zT0s/D8l1f2ryQeZHst5E+/wD/AGFfPOmfA/4r24eW78Wxzqhwlr/b8qb2/wDia5z4c6x4r/4XVdaB4j1u71N7IX0M0Ul7JPCZEjkB2c+oNfmvC+SU55xhp068Z8ko/wDpR9jnWbT+pzhUoyhzxPb7yT7ZPJPMiSSP/sV6h8I40j8OX+xPL/06T/0XHXldeqfCP/kW7v8A6/pP/Rcdf2bh/jPwyZ29FFFeoZhRRRQAUUUUAFT29u9w/wDzzRPvySfwUW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAooooAKKKKACiiigDS8P6P/AG3qKQf8sE/eTyf7FeqRxpbxxoieWifcjrl/h/p/2fSHuf47l/8AxytLxRrH9j6RJMn+vf5E/wB+vlMTOeIr8iOyH7uBk+KPGD2cklnY/wCvT78/9yuIjim1C6/5aXl2/wDwOSpNPs5tRuo7WH/WTV6ho+h2uhWvkxJ8/wDHJ/G9d0508DDkh8ZH8Q4W38D6vNH86Q2//XR6LzwPq8Me/ZDcf9c3rutQ8Q6fpb7Lm5SN/wDnn9+jT9d0/VPktrlJH/ufx1y/XMT8fQvkgeZ6fql74fuHSF3t5I/v28lejeH/ABJD4gh+T93On34Kfrnh+11yDY/7udPuT/3K82jkuvD+qf8APO7tX/77rT3MdD++R/DOv8eaH58H9oQp+/h/13+2lcJ0r1+zuIdU02OZP3kEyV5RqNn/AGZqN3a/88X2V0ZfWf8ABkFaBBRRRXtnMFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAW9H0t9Y1FLVH8vf9+T+4laUeof8Ihr06WM73dqnyPHJ/HVDQ9U/sPUUutnmJ9x0/wBir3iTSIIUTU7F/Msbp/8Avh68qrrW9nU+BmtP4C5eeE4Nb/03RZ4Y4JvvxyfwSVF42mga6sbWN/tE9smyaSubizH9x3j/AOudBzTp4WUJpynsHOLRRRXqGQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVPBC87/880T78kn8FJu24Bb273D/APPNE+/JJ/BUlxcJ5fkw/u4P/Q6Li4Ty/Jh/dwf+h1UrJJvVjCiiithBRRRQAUUUUAFFFFABRRRQAjVpeH9H/trUI4P+WCfPM/8AsVnV33w/0/7Po73X8dy//jlefja3saN0a04HURxpbRxoieWifcjrk/FHjB7SSSysf9en35/7la3inWP7I0d5k/17/In+/Xm1hZzahdR20P35q8fB0YVP3lTY6JzI445tQuv47u6f/gclbVv4H1e4j+dIbf8A66PXdaPoVrolr5MKfP8Axyf8tHo1DxBp+lyeXc3KRv8A88/v1pPHzm+Sghcn85wt54J1e3j37Ibj/rm9Zun6pe+H53SF3gdPvwSV6bp+uadqnyW10kj/APPP+Oo9c8P2ut2+xv3c6fcn/uUQxj+DEQHyfyEfh/xLDr0Pyfu50+/BWT470P7RB/aEKfv4f9d/tpXIRyXOg6p/curZ/wDvuvVLG4h1TTUmT95BMlZ1YfVaiqU9gh754/to6VY1Cz/szUbu2/54vsqu1fS05+0V0cYtFFFWIKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA9V+GX/Imx/9fU//AKMrnfjF/wAhLTf+vGf/ANGV0Xwy/wCRNj/6+p//AEZXO/GL/kJab/14z/8Aoyv4d4V/5OhW/wCvtf8A9uPUn/APj23/AOPVP9yvoTR/+QLpn/XrB/6Lr57t/wDj1T/cr6E0f/kC6Z/16wf+i6/s7D/GefMt0UUV6hmFFFFABRRRQAVb0fS31bUEtUfy9/35P7iVUq9omqf2LqCXWzen3Hj/ANisKvPyP2e44F+O/wD+ER16dLGd7y0T5Hjk/jq1d+FIdc/03RZoY4JvvxyfwPVTxJo8MEaanYv5ljdP/wB8PWHFmP7jvH/1zrzadP2kPaU5++dB0nje4hkurG1R/tE9qmyaSuaFLRXfQp+wp8hlMKKKK6jMKKKKACiiigAooooAKKKKACiiigAooooAKnt7d7h/+eaJ9+ST+Ci3t3uH/wCeaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MXyQf+h1VoorVLlAKKKKYBRRRQAUUUUAFFFFABRRRQBpeH9H/ALb1FIP+WCfvJ5P9ivVI40t40RE8tE+5HXL/AA/0/wCz6Q9z/Hcv/wCOVpeKNY/sfR5Jk/17/In+/XymJqSr1+Q7afuQOU+JnjR9PsZ7LTX/AH6ffuP7leKxx3WqX3yedeXb/wDbSSr3xek8SReCZP8AhFLWS91+S9tYEjFr9q+R3+eTZWT4f8SfEnw/Yxww6C8kn8dx/wAI6/mPX88eIWSTr5zDkn7nLH/24/T+Gc2o4XBThOHv8x6n4Ml+I/hXS3s9C8QNoFk83nPAgjk3v/q9/wDq5P7iVp3njb4x28e9PGNzef8AXvHB/wCzwV4L4m/aY8SeCb+Gw124s9M1CT94trPo379vqn362NL+M/xD1zzEs9LeSSP/AJZyeHZ0k/74eOvgf7FrU4cntv8AyY9z+2KFSpz+w/8AJSj+0B+1l8ZPhj4a0q70nxzqlldPqDwTJPaWp3ps3/8APCvoD4N/tGeMfHvw48Oalc+KLttRl0+Ga6Ro4N5f/np/q6+YPi/4X8YfGrw/HpuuaDexyWr+faz2miOkiP5f/j6f7FeNaD8QPjt8L3sNLs/CVyF0hfssedAZ96Ie77Pn+teh9SxFPCQgpx51L+Y876zh54qdb2On+E/RLx58S/iFBpv9oad4u1CB4f8AXxxxwfOn/fuvFPE3jLXfG18l94g1ObV7uGHyEuJ4449iff2fu4/9t6yrP4ufEbUPD8eoTaQ9vYvBvf7XoLpGkf8At184aj8WfjEuuanZaXoo1i0tLnyfOtNEWaP+/HyicHY6fpXi4zJcbiPf54cn+I9nB5tgsP7nsff/AMJ9LaX/AMhS0/67x/8Aoyvk/wDZKz/wszxLxj/iUzf+lUFacfxc+O9vIjp4RuA6PvH/ABTZ/wDiK4XwDa/FL4e63d6po/hDVvtlzA0D/aNGd49jOj/cKf7FdeAymeGw2Jp1Jx/eJfaMsdmccRiaNSEJ+5/dPuLw74i1fwjqkep6LqD6ZqSI6JPBHHI+x/v/AOsrqv8AhfnxK/6Ha/8A/AeD/wCN18Pav8cPjPoNm93qPh/+z7Jfka4uvD4VF9t7pRpfxw+M+v2aXeneH2v7JjhZ7Xw+GRv+BoleVDI8xpx9ytHk/wAR6FTOMuqz/eUJc/8AhPtfVP2gPidFpGqzx+N9TR4LK6kjk8i1+SRIHMf/ACzryL9mr9sH4z/EKz8TtrnxC1O9+xtbCH/R7X5N/nbz/q/9ivDrr4o/HG8t7m1fwle+XNA8D/8AFOtyjpsP/LP0rlfhtL8VvhfHqSaH4Q1MLfeWZlutEeb7m/bjKf7Zr16GAxdPBV6VSrHnly8vvHlVsZhqmMo1IUPchzfZP0E/4X98TP8AoddQ/wC/Fr/8brL8QfFjxn4v0mfTda8TXmp6dM6O9rPHBsfZJ5if6uP2r4+/4W18dv8AoT7r/wAJpv8A43WTb/tD/Fe71uTR7fSobjU0dkewi0gSToV++NmN/wBfpXmf2LmVRfxo/wDgR6H9rZfT/wCXH/kp798ZOPhH4yP/AFD/AP2pHXC/situ+Fd8cY/4nMn/AKJhrgfEHjz42eJdA1DR73wleC2vohDP5fh1o5Cnt8nFZfgHWvi/8O9Dl0rQ/COpfZJLn7SwuNCaZ9+1V7p/sV6FPKqlPLamE5487lf4jkqZlCeYQxXJPkUf5T63kjSSN43/ANW9egD49fEZenjG+H/bvB/8br4rT4rfHdm2L4QuPw8NMf8A2nWPov7RnxW8RX7W2m6bbarcRpuaGz0aOV1T/cQdK8qjkGOppunOP/gR6FfOcDVa9pTl/wCAn3b/AML8+JnT/hN9Q/78Qf8AxuvE9e/bK+NNh+0xp3hKL4ianHo015p6PB9ntf8AVyQxO4/1H+2a8Yj+LXx2aTB8IXIH/YtN/wDG64nUYfirqvxIt/GknhDVhr0M0EqFdEfyt8KIifJsx/AK9nLcBicPObr1Yu8P5vtHjY/GYavCHsKH2v5T9Bv+F/fEv/od7/8A78Qf/G6wPFXjzxF46NqfEes3Ormz3+T58ccezfs3/wCrj/2Er5K/4Wt8eP8AoUp//CcP/wARWZrn7Qfxb8MeV/bWjwaR5vMP23QxFv8A9zeleO8jzLEe57aL/wC3j2FmuXUff9hKH/bp9XV8q+Hc/wDDVPirPT7ZquPzkrQh+LnxykRGXwncMrHev/FOMR+HyVl/DHQ/F0/xeuPEfiTQNS003n2qaa4n0+SGAO8ch/uepNfacFZbUy/NITrzjq4/a/vHznE2YQx+D5KEJntdeqfCP/kW7v8A6/pP/RcdeV16p8I/+Rbu/wDr+k/9Fx1/XmH+M/HJnb0UUV6hmFFFFABRRRQBannTy/Ji+SD/ANDqrRRSS5QCiiimAUUUUAFFFFABRRRQAUUUUAFFFFAHqnhuPy9A03/rjG9c98R5P3mnQ/7710Phe4+0eHbF/wDph5dYPxIt/wB1Y3X9x3j/AO+//wBivk8P/vSv3OyfwFX4b26SXd9df8tERI0rZ8aaw+maeiWz+XPM/l+Z/crF+Hd4kd9dWz/8tk3p/wAAq58RLN5LO0uk+5C+x/8AgddFSKeNtUCn/DOIpkcn7zej/vE/551o+G44ZNesUuP3kDv/AORK7bxtp8E2hyTOnzw7Nj16NbFKnUVOxhCAvgvxA+sWskFy/wDpUP8Ay0/vpWb8RNP+SDUE/wBZ/qHrN+H8n/E+f/bgeuo8cJ/xTV1/wD/0OvKnH6vjPcOj+JAo/Du88zTp7X/ni/yVieNo/L8RSf7cCPWn8N4/l1J/9tP/AGeszx1J5niL/chRK6aP+/SsZ/YOfooor3zmCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAn0/T59Un8i2R5Pn+eSNPuV0+qT6N4fuk099MS4+T559nz0fbbrS/BVpNp37ve/wC+k2b9lMt7yHxna/YrzZb6qn+on/v18/UqTqvnn8COgzde0D+z4/tti/2jTpvuSf3Kff6nZx+G7XTLTfI7/PM8n8FVbfVL3RY77T/k8t/keOT59lZma7KdCdSzqPYjnFooor1DIKKKKACiiigAooooAKKKKACiiigAooooAKKKnghed/8Anmiffkk/gpN23ALe3e4f/nmiffkk/gqS4uE8vyYf3cH/AKHRcXCeX5MP7uD/ANDqpWSTerGFFFFbCCiiigAooooAKKKKACiiigAooooAT+KvVfDcfl6Bpv8A1xjevKv4q9V8L3H2jw7Yv/0w2V4mZ/BA6aBz3xHk/eabB/10eq/w4t0ku765/jRERKt/Ei3/AHNjdf3HeP8A77//AGKqfD28SO+urZ/+Wyb0/wCAVzR/3J8ofbNrxprD6XpyJbP5c8z+X5n9yvOq7f4iWbyWVpdJ/q4X2P8A8DrlvDkcMmvWKXH7yB3/APIldODcKdB1An8ZnRyfvN6P+8T/AJ516R4L8QPq1rJBcv8A6VD/AB/30o8cafBPorzOnzw7Nklc14Ak/wCJ8/8AtwvWdScMXhnUtqhQ/dzNP4h6f8sGoJ/rP9Q9WPh3eeZp89r/AM8X+SrvjiP/AIpu6/4B/wCh1lfDeP5dSf8A20/9nrmvz4J36M1+2ZnjiPy/EUn+3Cj1z9dB48k8zxF/uQIlc/XtYW/sIHPU+MKKKK7jIKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA9V+GX/Imx/9fU//AKMrnfjF/wAhLTf+vGf/ANGV0Xwy/wCRNj/6+p//AEZXO/GL/kJab/14z/8Aoyv4d4V/5OhW/wCvtf8A9uPUn/APj23/AOPVP9yvoTR/+QLpn/XrB/6Lr57t/wDj1T/cr6E0f/kC6Z/16wf+i6/s7D/GefMt0UUV6hmFFFFABU+n6fNqk/kWyPJ8/wA8kafcqCuu+23Wl+C7WbTv3Zd/30mzfsrgxNSdOyh1NKYuqXGi+H7pNPfTEuP3fzz7PnrI17QfsMf22xfz9Om+4/8AcrShvYfGVr9jvNkGqp/qZ/79Y9vql7oiX2n/ACbH/dvHJ+82VwUY1IvT4zUtahqdnH4btdMtN8jv88zyfwVh0UV69KkqSOcKKKK3EFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFT29u9w/wDzzRPvySfwUW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAooooAKKKKACiiigAooooA9U8Nx+XoGm/8AXGN6574jyfvNOh/33rofC9x9o8O2L/8ATDy6wfiRb/urG6/uO8f/AH3/APsV8nh/96V+52T+Ax/Cdukkd/P/AMtE2IlUPiB4gn0PR40tn8u7un8tJP7lZ9/42g8CeGdS1CW1+2J59rGlv5/l798nl14j8aP2lbSz0Wwv18LXMscMxRyuqJ8m/wCkdfgfiRgcZiM2nKhH7Mf/AG4/SOGcZhaGF5a8vtng37SjEfG3wg7N/wAuVt/6VT19TahJ/wATW7dHfzEnk/ef9tK+Q7r4qaV8Uvjf4R1i60mTT7CzWG3+yT3fn72SR3TL7E7uK+p/i1+0dpC+F77U28LzySW0m+P/AImKf89P+udfnOY5TjMRhcOqcffgj6PBZvg8Pia3PL3JyPZvh/4ofXLGSC5fzLu1/wCWn9+Osn4qaX+7tNQT/Wf6h6+b/gz+1xp+q+MXhXwhdQD7K+8/2on/AD0j/wCmFeh/FP8AaYsrDwDqV0/he6kjTy/+Yin/AD0j/wCmdcX9j5hUpck4f+knb/auX063PCf/AKUdnb6hJcfB3xnau/8AqbWTZ/wPy68z8BwfY7TWXT/l5vfPf/vxGn/slYfw1/aKtfH3g/xfYw+G7qwS+jjtftD30cnl/wDkNK4XXf2iLH4Y6veaLPoc2ryCSOffHepBs+T7n+rkruWV47+yp4dQ9/mPO/tPBf2t7bn9zlPe6K+df+GztL/6Eu6/8G6f/GKP+GzNL/6Eq6/8G6f/ABivm/8AV/NP+fX/AKSfULPMs/5+/wDpR3H7TUh/4Ulq2P8An9tf/Q5Kn/ZrOfgloJP/AD2uv/R1eM/FP9pKy+JHgK70GDw7NpTTTwTG4k1BZ/uf7HlpTvhl+0rZ/DzwHpuhT+HZtSa1eZvtEWoLBu3vv+4Y3r3f7Ix39lfVuX3+e+6/lPD/ALUwf9p+35vc5f8A24+r6K+df+GzNL/6Eq6/8G6f/GKB+2bpZ/5ku6/8G6f/ABivB/1fzT/n1+MT3f7byz/n7/6UfRVfMHw8BH7ZHiI9v7Q1j/0XPWv/AMNlaXux/wAIXdf+DdP/AIxXlnh/4wW2g/GrUfHD6TNcQXVxeT/YftYR188OMeZ5fbzP7lfQ5XlGNw1HEwqx1nHTVHz2YZpg8RWozpy+CR9t0V87f8NmaZ/0JV1/4Nk/+MUn/DZul/8AQl3X/g3T/wCMV8//AKv5n/z6/GJ9D/beWf8AP3/0o+kNL/5Clp/13j/9GV8mfsjA/wDCy/Evp/ZUv/pVBXU2f7aWlW93DN/whl1+7ff/AMhdOf8AyBXkPwc+LEXwr8Tapq1xpj6ml5aNa+RFdeRt3So/39j/ANz0719FluT43D4bFQqx1mo2PnswzbB4jE4adOekOY+4qK+df+GztL/6Eu6/8G6f/GKP+GztL/6Eu6/8G6f/ABivnf8AV/NP+fX/AKSe/wD23lf/AD+/9KPoqvm79spisPgnHpef+0ast+2Vpakj/hC7nj/qLp/8j15f8b/jNa/FpdC8jRZtI+wGYsJLoXHmb9nfy0/uV7OS5PjsJmEa1eHuf9u/yni5tmuCxGDlRoy9/wD+2PtDT5P+JXpv/Xja/wDoiOqniP8A5A8n/XSOvBrf9sbT7e1tYD4Nun8iCODedUTPyR7P+eFb3hf9oWz+JmqRaBbeG5tMeRHn8+TUEn+4hf7nlp6V28N5LmNHPMJVnDTnj/L/ADFZpnGX1svq04T15JfzHV16p8I/+Rbu/wDr+k/9Fx15XXqnwj/5Fu7/AOv6T/0XHX9r4f4z8BmdvRRRXqGYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB3Xw71DzLGeyf78L70/3K6HWdLTWNOntX/j+5J/cry7S799HvY7qH+D78f99K9V0/UINUtY7q2fzI3r5TGUXRre0gdkJ88DyeOS50fUf+fe7tXrvdP8YaZrFr5F55cDv8jxz/AHHqz4g8Lw658+/7PdJ/y0riLzwnqdn9+2+0J/z0g+euvnoYuC53aZHvwOws9D0HR5vtqSQx7PueZP8Au0rnfFnihNX/ANGs/wDj0T53k/v1ixeH9Rkk+XTLr/tonl10Wj+AJ7iTfqL/AGeP/nhH9+jlo0J+0nPmF75P8O9Lf9/qD/6t/wBwlT/ETUEj0+C1/jmff/wBK6W5uLXR7He+y3ghSvNTI/i7xInnfu0d/wDV/wBxK56X76t7eeyNfghyHaeC7P8AsvQEkm/d+d+/evPdQvP7U1G6uv8Ans//AI5XX+NvECR2/wDZls/zv/r/AC/4I/7lcQrZzXfgoybnXn1M5/yC0UUV7JzCfeo+7R92uf8AHHjjTvAej/bbz95PJ+7tbSN/3k8n/wAR/fkrkr14YeHPU+EDoN1LXEfDP4lwePLH7Lc7LPXoY/MntI/9XNH/AM9I/wDPyV2y1nhcVSxVL2tEIT9oLRRRXcAm6lrNs9f0vUNcv9Itr2GTUtP8t7q3/wCWiVpVlTqQqfAAUVaggTyfOuPufwR/36q1qncAooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFEcb3D7IUeT/rmm+k3YC3o+kza3fR2sX7v+N5P7lbv/CN6LcT/YrfV3+3fc/efc31T8FXyWurvHM/l+cmzzP+mlQyeC9UjuvsqQeZH/BPv/d149apJ1WufkR0w+Am0/ULnwjqL2d2nmWr/fg/v/7aVR1q3srfUP8AiXTb4H+dP9ir2talNJa/2ZqMKSX1s/8Ax9RvWHWuHp3/AHj/AOHM6gUUUV6hkSR27z79n7zYm+o6I5Hjk3o/lulXpI0vkeaJPLnT76f+zpWLbT1Ao0UUVsAUUUUAFFFFABRRRQAUUUUAFFFTwQvO/wDzzRPvySfwUm7bgFvbvcP/AM80T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVSskm9WMKKKK2EFFFFABRRRQAUUUUAFFFFABRRRQAUdan0/TZtUuvstpH5k+zzPL37KLzTp9LupLW4j8udPvx799cnt6XP7Hn98fJ9sgruvh3qG+ynsn+/C+9P9yuFq3pmoPpF7HdQ/wffj/vpUYuj7enyIunPkPUNa0tNX06e1f+P7kn9yvK45LnSNQ/597u1evV9P1CDVLWO5t38xHrN8QeF4db+ff9nuk/5aV4OGrewbpz2OmcPaEGn+MNL1e18i82W7v+7eOf7j0Wei6DpE/wBtR4Y9n3PMn/dpXH3fhPU7P79t9oT/AJ6QfPVWPQNQkk+TTLn/ALaJ5ddn1ah9ioZ80zZ8V+KE1j/RbP8A49E+d5P79X/h3pb/AL/UH+4/7hKg0fwBPcvv1F/s8f8Azwj+/XZ3Fxa6PY+Y+y3ghSsK1aEafsKBcI/bmc18RL5I9Pgtv+Wkz7/+AJWh4Ls/7L8PxvN+7879+9cX5j+LvE0fnfu43f8A1f8AcSug8ceIEjt/7Mtn+d/9f5f8Cf3KJUZqEMMHP9s5HULz+1NRu7r/AJ7P/wCQ6rDigGhq+kpwVNciOQWiiirEFFFFABRRRQAUUUUAFFFWoIE8nzrj7n8Ef9+k3YCrRRRTAKKKKACiiigAooooAKKKKAPVfhl/yJsf/X1P/wCjK534xf8AIS03/rxn/wDRldF8Mv8AkTY/+vqf/wBGVzvxi/5CWm/9eM//AKMr+HeFf+ToVv8Ar7X/APbj1J/wD49t/wDj1T/cr6E0f/kC6Z/16wf+i6+e7f8A49U/3K+hNH/5Aumf9esH/ouv7Ow/xnnzLdFFFeoZgKtaPo82t3yWsP7v+N5P7lVY43nbZCjyf9c031u+CdQSz1iRJn8vzk8vzP8AppXJWny058m5rAtnw3os8/2K21d/t3+r/efc31T0/ULrwjqL2d2nmWr/AH4P7/8AtpUMngvVI7r7KkHmR/wXG/8Ad1NrWpTPa/2ZqMKSX1s//H3G9ebG9R8nPzpmhR1q3srfUP8AiXTeZA/7xP8AYqjSNQvSvXpw9nCxzi1JHA8+/Z+82JvqOiOR45N6P5bpWrv0EFFXZI0v0eaJPLnT76f+zpVKlGXMAUUUVQBRRRQAUUUUAFFFFABRRRQAVPb273D/APPNE+/JJ/BRb273D/8APNE+/JJ/BUlxcJ5fkw/u4P8A0OsW3sgEnnTy/Ji+SD/0OqtFFapcoBRRRTAKKKKACiiigAooooAKKKKACiiigAooooA7r4d6h5ljPZP9+F96f7ldDrOlprGnT2z/AMf3JP7leXaXfvo97HdQ/wAH34/76V6rp+oQapax3Vs/mRvXymMouhW9pA7IT54HhviaFrO3kS4ggkktp45PInRJI/MT/YetDS9U8G+JLGS1v9E0W3kdPLe3u9Og8t//ACHXXfF/T9Hh8Mz6nqV1/Zex0g+17Hk+/J8n7tK+ctU8WaJpf39Qe4T/AJ6WljdSf+06/mzxEqYytnEKtCEpe5H/ANKkfrHC/wBS/s/krzhCfP8A/Inrlh8MPhvoF1/aMPh/w5BJH+8ST7FB8n/kOuZ8eXmg+II/7OttB0iSwR/MeSTToPn/APIdedw/Ebw9NJ5az6h5n+3pF6n/AKGldXoeqeFJJI31XXvs8H/PvHaz+Y//AJDr8zdPNKr5Y0pw/wDAj6+Esup+/OtD/wAlOp+FHw90KD7Xqi+G9Gg3p5EciaXAm/8A56f8s6+UP2yNYh0vUNc0WHbH9p1ify4Io9iJCknH/tOvsG++OXw98M6O9zca6lpY2yYA+wz/APxuvz48da0nxw/aH1O5t5TNpF7qM8kOVZP9G8wv06jf/N6+xyPD4ilPlqs+MzytQrQ56Z6F8G9AHhj4d2r3H7trr/TpvM/g/wAp5dfOHjTXW8TeKdT1MhglxNlO2E6J/wCOYr2f47fEaHS9Nk8M6cyfaZk8u62f8sU/5518819ifHk8a/OuRhM1+hun+DfDn9lWDv4Y0OSR7WCTzJNItf8AnnH/ANM6/PWNcMG2EoPevuG1/aE+HMOm2CyeJ443S1gR8afdH50RB/c9jXxfE1LF1FR+qxl12+R9zw3Vw1N1frEo/M7H/hDvDn/Qs+H/APwUWv8A8bo/4Q7w5/0LPh//AMFFr/8AG65D/hof4a/9DVH/AOC+6/8AiKP+Gh/hr/0NUf8A4L7r/wCIr4H6rmn8s/8AwGR9z9Yy3+aH/gUTpNW8H+HIdF1mSPwzocUken3UiSR6Xa/J+4f/AKZ14T+yHoul6tZ+MVv9M0/U1jeyMf262SfZnz+UDqcHivTdQ/aA+Hl3o+pwweJo5Hlsp4FjOn3Q3u8Dxp/B7ivG/wBmj4ieF/ANr4lTX9T/ALMN9JamAmKaTeqCbeP3anH30r6LC4fMP7NxMKkZ8/u27/F0Pn8VWwX9oYacJw5PePpj/hDvDn/Qs+H/APwUWv8A8bo/4Q3w5/0LHh//AMFFr/8AG65D/hob4a/9DVH/AOC+6/8AiKP+Ghvhr/0NUf8A4L7r/wCIr536tmn8s/8AwGR9B9Yy3+aH/gUTrv8AhDfDn/QseH//AAUWv/xuvlv9rPS7HSvHmkQWFjZ2Eb6Uknk2NslvHuM8+TsTiveB+0L8OT08VR/+C+6/+Ir54/aS8beHvHXjLTr3Qr038EGmpA8nkvH+88yZ+j89HWvp+HqOPp4xyxUZ2t1Pm8/rYWphFDDygeL0Vu2eiwQWj3+oyNFbfdhjT78z/wCz/sf7dYVfpZ+Yjsc13/wPsob74teEYbmCK5t5dTt0eGePzEdTIMgrXAd67b4Q61Z+HfiX4c1LUp/sun2WoQXM0+1n2IjgnhOT07Vhied4efL2/Q7cJye3hz7H3J/wh3hz/oWfD/8A4KLX/wCN0f8ACHeHP+hZ8P8A/gotf/jdch/w0R8N/wDoa4//AAX3X/xFH/DQ3w1/6GqP/wAF91/8RX4q8Lml/hn/AOAyP2RYjLrfFD/wKJ1//CHeHP8AoWfD/wD4KLX/AON181eFYYbX9qLxLDbW6wW8NzqiLFDHsREHmAAJ6Y7V7OP2hvhweniuP/wX3X/xFeJ+B9Wstf8A2ktc1TTZvPsL2bUZoJ8Mu9H8wp9/kdR1r9G4Fo4ynmkPrMJ/Z3/xHxHFdbC1MH/s84HuFeqfCP8A5Fu7/wCv6T/0XHXldeqfCP8A5Fu7/wCv6T/0XHX9f4f4z8ZmdvRRRXqGYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAmKuaPrF1oc++2f7/wB+OT/VvVMGg1jOmqi5JjPRtL8eafefJc/6HP8A9NPuf991u299Bef6qZJP9x68cGaZ5cf9yvKqZZH/AJds6Pbns1xdQW/+umRP+uj1hap440+z/wBS/wBsf/nnB/8AF15t5cf9yn80QyyP/Lxh7cv6xrl1rc++5f8Adp9y3j/1aVUike3ffC7xyf8APSOmUmK9WFOFOHIjnFrjviN8RrX4e6b8iJea1On+i2n/ALUk/wBj/wBDrsO9c5+zZ+zJYfGT47eJE8aeI9P1yx0KRLu602ObN3qe/wD1aSR/wQR/cf8A79/cr5biHMqmXYa9H7Qck6n7uB558J/jP/aE0ekeJL1JHmf/AEXU/kjj8z/nnJ/7JJXs2Atezftf/sWWPxe0ubxR4IsINM8cWUCQyWa7IbfWYEQokLn7iTImUjkPGMRv8mx4/jP4V/FiaxuR4V8WedaXVtN9hjuL5Nk8M6fJ9kut/wBx/wCD/wBDr53IOIr/AOz4qQTpzwsuSrsexyb9kmzZJJs+TzH8uOuM+BX7MXjH9pX4j3+q+NorvRvDelzi31CeMbJJnQh/sNr6RjOTJ6Of43+Sf4p+OLr4f+GZL22snknkfyEuJI/3FrJ/z0kr2n/gnR8J/Hmk/wBq/ELW9W1DTfD+vwhrfR7lSX1NjjOozb/uZ/gP3nT/AGBGKji7GxtCjTHQhCvWhCZ57+19+xpd/CG+PxE+GMNymgRyJJe6VZAvPpL/AHPPg/vwf34/4P8Arn/q8uPW73Q/C0F94nghs9SRP9Kt7T5/3/8AzzT/AG6+pv25vgv4x+NXwjgt/CuqzEaZObu68NxhETWkA+RDIf44z86Ifkd+vOx0/N3T/FmseJP9F1h7q4k0xPI8y4Ty5E/56Ryf7f8A4/8Au/nrzuFcb7Gc6c/tG2Koww9bkh9s+h/DfiC18UabHdW37v8A5Zz28n+sgf8A551578XPjAnhOOfSNHnT+1v+Xq7/AOfL/wC3/wDoFebah44uvB8k76bevZ3ezy57iP8Agj/+Lr7T/Yn/AGKW8PtYfEf4j2DjWsi50Xw/dJzp56i6uv8Ap6/uR/8ALAf7f+r9nPOIFQh9WofGYU6c68/ZUtj5j1j9l74r/DPwP4b+IkumXFlLe3SEWivi/wBNeaTy4TdRv2nMnzp/B5mx+r7PaNJt7qPS7e61dIY7jy08+O0f928n8ez/AGK9H/4KReGbTUtH8Harqvjg6RYW2o+Qnhm6Yi3vZH63abPn3wJ993JRIy5TY5+fw7xh44/4RfToLX7V/amtPBHskuNn3P8AnvJsrm4RrVJKtc0qUYUJ8kDrLi7+1zSb3TzE/wCWcf8ABUdeDaP4k1HQ9Zk1GF5Li7mf9/HJ8/2r/rpXusEjyW8cjwPbu6fPBJ/rE/2K/S6M+ZBqSUUUV1GQUUUUAFFFFABRRRQAUUUUAFFFFACZFGRVvTdJudWjn+zoknkpv8vf9+qUn3JKx9pTvYYkn+rr0SzuE8P3WjaTbxp5d0kjvP8A8ArnPF+l2WlvY2tvD5b7Pnk/v0631/S7zS4LPXZPs8cPyQ3W/wAuvIxDdeHtF8B0U/cKV3p82qeIr61tk8x3mf8A9GUSaxq2n77J72aPZ8nl76w/i58WrH4S/CfxFq/hF9Pv9VtoY3T7VOHjfzJ44/uJJvf/AFlfFv8Aw3T473Of+Eb8OSHO/wAzyL3/AOSK8PEZ5hcHP2eJOiGFqT1gfbZbBo7V4v8As1/HLU/jPpfiW612DRdHfT57VII7SR4d+9JN/wDrpH/55x17LHcQXD7Ibq2uJP8AnnBOklfSYLHUMdSjVos46lOdOfISUUUV6pkFEcjxyb0fy3SiigC9JGl+jzRJ5c6ffT/2olUaI5Hjfej+W6VekjS/jkmiTy50+/H/AO1ErD+EMo0UUVuIKKKKACiiigAooqeCF53/AOeaJ9+ST+Ck3bcAt7d7h/8Anmiffkk/gqS4uE8vyYf3cH/odFxcJ5fkw/u4P/Q6qVkk3qxhRRRWwgooooAKKKKACiiigAooooAKKKKAE6ZNV7zULWzhj+1TJb+Z9zzHqx6V5vrHiCfXI40mjhj8mTf+7r834x4qocL4WMn/ABZ/D/26foPBfClbirFcn/LmHx/9vc3L/wCkmrb/ABC1vQtZnutOurXCPJGkklqknyUv/CeavrOuJPqV1a7JpE8+TyI44/8AV1D8P/DcPjDxRbaTczXNrBJFNNvhT5/kT/bqDxtoMPhnxZq2kW0s08FrIgSeZPnf5Ef+D/fr+Qo8YZtHMPr3tpb/AM0vh5vh/wAJ/VsuFuH5w/sr6tDn5Pi5Y/4eb/EdxZ6hBfR77WaO4j+55kb76nPrXn2h+IJ9LjS1hjhkSaePzPM/74rvo5Y5DhHST/rm9f2NwhxRR4mwXtofHD4j+TeLeF63DGN9jP4J/AXtH1i50WbfbP8Af+/HJ9x67rS/Hmn3X7u5/wBEn/6af6v/AL7rzuivsK+Ep4jXqfDwqezPY7e8gvP9VMkn/XN6Li4gt/8AXTJH/wBdHrxjy0/uUeWn9yuD+y3/ADmvtj0nVPHGn2n+pf7W/wDzzg/+LriNY1u51qffcv8AIn3II/8AVpVDJo213UMFTo69TOc/aEqSPA++F3jf/npHUdFFd9jIKKKKYBRRRQAUUUUAFFFFABRRVqCFI0+03H+o/gT+/Ut2AW3gSNPtNx/qP4I/79QXFw9xJvf/APYouLh7iTe//wCxUdZpO92AUUUVsAUUUUAFFFFABRRRQAUUUUAeq/DL/kTY/wDr6n/9GVzvxi/5CWm/9eM//oyui+GX/Imx/wDX1P8A+jK534xf8hLTf+vGf/0ZX8O8K/8AJ0K3/X2v/wC3HqT/AIB8e2//AB6p/uV9CaP/AMgXTP8Ar1g/9F1892//AB6p/uV9JeGdJudX0O3+zoknk2UD+Xv+/wDu6/s6lNU9ZnnyGVHJ/q6WT7klb/i7S7LTHsbW2h8t9nzyf366alX31T7j1OktLlPD11o2k20aeXdJI7z/APAK5G8sZ9U8RX1tbp5kjzP/AOjKvWeuafqGnQWWtI/7n/UXaVYk8QafodjJa6Fvknf793IleRTjUo1NI+9/WpoZMmr6tYb7J72ZNnyeXvrNo/3/AN5JRXuU6apnOFFFFbCCiiigAjkeOTej+W6VekjS+R5ok8udPvp/7OlUaI5Hjk3o/lulYuLeoBRV2SNL9HmiTy50++n/ALOlUq0jLmAKKKKoAooooAKKKKACp7e3e4f/AJ5on35JP4KLe3e4f/nmiffkk/gqS4uE8vyYf3cH/odYtvZAJPOnl+TF8kH/AKHVWiitUuUAooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACYq5o+sXWhz77Z/v8A345P9W9UwaDWM6aqLkmM6LxZrmg+OPCr6ZrfnWdu88DvHHP5fzpJ/wA9K5O3+C/gO8j3w/bbiP8A6Z6o8lUPE0iRaS7u6Rx70/eSPsrifMtf+fqy/wDAtP8A45X8m+JGMq5dnHsaC9zkj/6VI/YeF8Bh8bl/PUf2/wD5EofFTw/oPgfxppOl2z/Y7C6gtZH+13X7z555I5P3ldn4g8F/DbS5J44YNU1CdH8vy7fUX8v/AL7r5A/aR8ofGrwh5ckMkf2O1+eN1dP+PqSvqbULi1+3Xf8Aptl/r5P+XpP+en/XSvhMyzPF0sNhp05/Gexl+V4SpisTCpD4DhvFHwb8I+MLjzdQg1byE+5brqbmNPzjr438eWv/AAgfxO8R2OhzXNhDp+o3NrbvHMfMVEkdPvj2r76+0Q/8/wBZf+BSf/HK+CfjUyyfFnxmysrr/bF3ynT/AFz11cMYzFYqtV+sT5rHNxFhMLRo0/YQ5Th5JGlkZ3be7dTTKKK/QT8+JoxuwM/eNfa9r+zP8PZ9NtHksNT3vbQSP/xMB/GiH/nn7mvieL/Wqe2a/STT7i1/srTf9Ntf+PGD/l6T/nhH/wBNK+J4pxOIw8aP1efJv+h9vw1h8PXdb28OY+f/AI3fA3wf4K+Gl/rOk2l7Few3MEaST3okT5m+f5Ngp/wV+BPg3xl8MtL1nV7O9nvbmSdGkgvdifI/9zZ6V2f7S1xDJ8FtVSO6tpJPttr+7jnST+OSrH7NtxDH8F9CSS6to5POuf3ck6R/8tq8H65jP7G9pzz5+f5/D+R7v1PCf2t7Pk9zk/8Abiv/AMMv/Df/AKB+p/8Agw/+10n/AAy/8N/+gfqf/gw/+116n9oh/wCf6y/8Ck/+OUfaIf8An+sv/ApP/jlfN/2lmH/P6f8A4Ee9/Z2A/wCfUDy8fsx/DkdLDU//AAYD/wCN14d4P+HOg6r+0Zq/hK4t7iXQ7O61GGONbgLJshSQp85/3K+wftEP/P8AWX/gUn/xyvmP4fSKv7YXiBjNAE/tDWD5kkibP9XP/HX1GU47G1aGKdWo7qGm587meDwlOthlThvLU9LH7Mfw5HSw1T/wYf8A2uvFfjv8NfDnw38WWIs7e5g0yTTkm+y3Fzve4m8yROHH8HyJX2TZx2scH2q8urXyP4I/tSfv6+Rv2y7/AO3/ABJ0eUvBJjRoU/cOrov76bjir4exuNxOLdPETdic/wAHhKGD56EDwrVNUm1a686bj+BI4/uInZErOoor9NPzEeflJFdf8KvD9p4q+JHhvStRV3sL7UILaZI22uUdwDg/jXH9ea9C+BEqRfF/wc7usaLq9rl3bgfvRWOJk44eco9v0O3BxU68IzPpsfsy/DsZxp+p89f+JgP/AI3Xjn7Sfwu8N/Dv/hHP7Ct7i2F6bkTCe5E33DHs6dPv19ax3EHl/wDH7Zf+BSf/AByvm/8AbIkSSHwYUmhkAF5/q5kk/wCePpX5hkeYYyvmEKdWc7a77fCfpGb4LCUMBKdOGv8A9sd3afsy/DyWytXew1PzHtoJH/4mH99E/wCmfuavWnwU8H+AZP7Y0W1v7e9Q+Skk94HTY/7t/k8v3rv9PuIP7K03/TbL/jxg/wCXpP8AnhH/ANNKqeJJIJNHkRLq1kk8xP3cc6SV1cM4/HTz3CU51p254/8ApReaYHAwyurOEIfBI5OvVPhH/wAi3d/9f0n/AKLjryuvVPhH/wAi3d/9f0n/AKLjr+38P8Z/Pczt6KKK9QzCiiigBMijIq3puk3OrRz/AGdEk8lN/l7/AL9UpPuSVj7Snewx9FFFbCCiiigAooooAKKKfJA8ao/8D0gGUUUUwCiiigAooooAKKKKACiiigA6Um6hqo65r+neF9Nk1HVb2OztEeOPzJP771jUqQpx55gXt1LWbqGv6Xpeq6bp1zew29/qn/HrHv8A9f8A5/8AH60qVOrTqfAwEj+9VH9hex+FQ/aYv7hNZ1BteiSY+GIJm2Wk29H+1FJPvzPs8zZHJ/Bvk+d/nSfUNQtdH06e+v50s7S1Te9xJ/BVr9gf4jadq37RXia3sPh/tfWrPzP+EgtU/f6dHGORP/Akc78/u/8AloP+Wgy6fnnGVSHsYQ+2a0OT28D9Jq+Sf2yv2Nrf43WM3i/whbw2vxAt4Nkkcn7uDWYE/wCWEx6JKOiSe2x8p9z637Vj6X4q0nWNU1DTLO9invdPbZcwL1jPvX5JCbpu6Po61GFeHJM/FRviR4rXw3P4Hltro6r9q/s3y3tXk1JH8zZ9h8j+/v8ALT+//BX7AfBBvHR+Fvh5viYlinjX7Of7R/sz/VdTs34+TzNmzf5fyeZv2fJihvgn4Jb4uJ8SG8OWv/CaLZ/YRrGfn2Yxv2/d8zZ+78zG/Z8mdnFX/HnxEsvAml+dM32q9mGLazjf55D/AET/AG67MVjJ4vk5zgw2F+qc85zNzX11I6HqP9k/Z/7X8l/sX27f5Hnbfk8zZ82zd1xzjOO1fmDafsx/H74pTfEjxN4hsUg8RWtz92+jEb6tdxmMOLH+DyfJjQI7/I+Ej/vun6D/AAr+LEfjGMaZqnk2muRfwKSqXKf89I//AImvTcCs8PiquFnz0jonRhivfPhH9iX9id9BfTviP8R9PaPWfkutF8OXic6fx+7urpD/AMvX9yP/AJYf9dP9X93j6cVzvi/xhp/gfR3v758/wQwJ/rJn7Ig9a8/+Gvxn/t7Vn0zW1isprmY/YWjPycn/AFEh/vjs/R/y34VJuo7sujCjhf3MD5r/AOCmMngGOz8KxalY3j/ERuNPvLI7Cun7x56Tu4w6HJ2J9/f/ALBk3/Hdn5ElrH9j2eRs+Ty6/Wj48fCnwb8Wvh3qOj+M4o4NNt45LlNUDLFJpjov/Hykj/cKcnJ+TZkPlCRX5F6LY2VjcarDpusrr1il08EN9DA8CXKI/wAk+x/nTelfccLYr2db2J5GKhOnief+Y3tHjvZNUtE01/LvvPj8iTf5f7yvfLfzvIj+0yJJcbPnkt02R76+dJLzTree0TUr37HBM8cb3GzzNkf/AD08uu++JfxWtfBej/2Xo2oJeat5Hz38k6Tx2qf89Hf+N6/SnmGFwsJznP4TjnOED1DmjOa898f3fxZ8GeFfAWsXfgefS4dc6yeXv+1Sf8s4HT79r5ifP8/z/wDfD139vI8kKSTQfZ5HT57fzN+x/wDnnvrfL82w2Y8/sfsgSUUUV7YgooooAKKKKACipII0kfY7+Wn/AD0ongeB9j/fpXV7AR0UUUwL2gW+oSX0k+lp5k8P362by1h8XWL6hpyeXfJ/rrX+/S+E99xo+s2Vs/l3zp5iVm+F7e9TX7TyYZo3R/3/AMn/ACzrw6km6k5reB0F/R9Qg8SWMej6r+7nT5LWevg79qb9oD+2YfE/w1l8L/ZJ9L1sp/aQ1Hz0fyHkT7nl/wAf+/X1T+0B8XPD3wX1CPUdYttQu7TULryEOmxpI8bokbv990r41+Bnxy8DeAv2oNY+IPizRbzW/C15Nqcseniytruc+eH8nfHM3l8b+efzr4HiXM/Y0fZYWfx/FE9HBUeeV5n07/wT3/Yhu7yL4ZfHuPxZ+7W5urr+w10l5D+5nntdnniTqdm/7leu/t7/ALbEnwPm1n4T3Xg281ZvE3hST/ibNrT2pg+1JPAT5Hkvv2bP79fnb+0b8fLf4gfGjxH4h+HU2seFPB92YDY6T5i2Rh2W8aSHyYH8tN7o7/J/frx3WNb1LXroXGpajcajOq7PMupnkfb6ZavyL6vOpU9tUPofaqEOSBlKfavXP2ffjdB8Edf1bU5NDbXPt9l9hES3v2Xy/wB5G+/Plvz8leSDrSrnnFe3h8RPCzVWk7SOKUOdcrP1k8K66PFfhPQ9a8n7J/adlBfeRv8AM2b49+zfWtXzz8Gv2nvCuo6H4X8JW+m642pWGkwwTSbYfILwQfOU+fODs717voesR6/pNvqMKPHHN5myOT/Wf6zZX9CZdjKeMoQqQnzHzFSn7OoX6KKK9gxCiOR45N6P5bpRRQBekjS/R5ok8udPvp/7USqNEcjxvvR/LdKvSRpfxyTRJ5c6ffj/APaiVh/CGUaKKK3EFFFTwQvO/wDzzRPvySfwUm7bgFvbvcP/AM80T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVSskm9WMKKKK2EFFFFABRRRQAUUUUAFFFFABRRRQAUU+S3eNEf+B6guLj7PaTzv/wAs08ysKlSFODqT2RrTpzq1PZwMnX/EQ0Oa3j+y/aPOSST7+yvPTIked7+Xz/y0rW8Ta7BrU1pJCk0fkI/+srtfgLZw3niLWBcW8U0a2KOnnRI3/LY/36/hzxC4glnGaTUK3NRp/B/29GPMf2lwZlEeF8h+t1qPLXn8f/bspcv/AJLI9c+Hszf8K78MJ5jOrabCfvf7FV/ipclfhj4pV5PlW067v9tK7jwv4ak17VINNsvs8EnlO0fmfImxf9yo/Efh5tC1W70q+8md4vLD/JvR/k3/AMdfk3sqnL9b5PcPnI42ksbZfGvf/wDJj4oYJNbvGkiHf8maXw/4gj+HcN3/AKD/AGh9q+f92/kbNn/7degfHC1htvHZjhgit4/sMOI4VRE++/8AcrzPVtNkuvL8kodnmcyV+u+HvEM8nzOFOdbkw0/i/wDAXy+fxHv8cZLDiTJPr1Cjz4lcvL/29KPN5fCe1/vPLR3R499IteZ/CO4e41XUt87yf6L/AMtH8z/lpXpnRq/uCjUVSF0fxc1ZtMWiiitiQooooAKKKKACiiigAooooAKKKKACiirUEKRp9puP9R/An9+pbsAtvAkafabj/UfwR/36guLh7iTe/wD+xRcXD3Em9/8A9io6zSd7sAooorYAooooAKKKKACiiigAooooAKKKkgjSR9jv5af89KQHqPwy/wCRNj/6+p//AEZXO/GL/kJab/14z/8Aoyul+HNu9v4RiR/v/ap//Rlc18Yv+Qlpv/XjP/6Mr+HeFf8Ak6Fb/r7X/wDbj1J/wT49t/8Aj1T/AHK+lvBtvqElnYTaWn7+Gyg3/wDfuvmm3/49U/3K+rPh5vuPB89nbv5d89jBIn/fuv7GnPkpnGi1eWkHi6xfUNOTy75P9daf36j0y/h8RWKaTqn7udP+PWeqXhi3vU1608mGaN0f9/8AJ/yzqPxPGg8RX3k/6vf/AORKXs71PY39B/3ypdWr2N1PbS7N8L7P3dVz2paQ9q9yG2pzi0UUVQgooooAKKKKACiiigAjkeOTej+W6VekjS+R5ok8udPvp/7OlUaI5Hjk3o/lulYuLeoBRV2SNL9HmiTy50++n/s6VSrSMuYAoooqgCp7e3e4f/nmiffkk/got7d7h/8Anmiffkk/gqS4uE8vyYf3cH/odYtvZAJPOnl+TF8kH/odVaKK1S5QCiiimAUUUUAFFFFABRRRQAUUUUAFFFPkgeNUf+B6QDKKKKYBRRRQAUUUUAFFFFAHB/HLwV/wsL4b3ugi9Sw866gn8+SEvs2Sf3BXy/8A8MaLuB/4TOP/AMFr/wDxdfY3iT/kFSVxFfyX4n5tjMvzuFOhLeEei/mkfsnCeVYPF5fz1o/a/wDkT5tP7GAY5PjSM/8AcNf/AOLpf+GMe3/CbJj/ALBz/wDxdfSNFfkX+sOaf8/fwifaf2Dlv8v/AKUfOA/YzULj/hM4/wDwXP8A/F14F418N/8ACI+Lta0P7T9qTTb2a0M2zYH2OU3be2cV+hlfBPxpwPi54zz0/tq8/wDR719hw3mmKzCpUjiJbeR8nxBluGwNKEqMThaKKK+5PgyaJTIwQHivpiD9jhbm0t5j4zRBNDHOP+Jc/R494H3/AEr5og5mTHFfpNp//IJ0z/ryg/8AREdfIcR5hicvhTdCW9/0PtOHcBh8c5qtE+cf+GMR/wBDrH/4LX/+Lo/4YxA/5nWP/wAFr/8AxdfSNFfC/wCsOaf8/fwifaf2Dln8v/pR81Xv7HpsbC+u/wDhLo5Ba200/lixcF/LjkcjO/8A2K8++CnwX/4W1BrT/wBsppP9neTzJbmYPv39h/ufrX2Zq/8AyANZ/wCwddf+iJK+ff2M8/Y/G3+/Zf8AtevewudY2pl2Jrzl78OXou54eKyrCU8wo0YR9yfMRf8ADGP/AFOyf+C1/wD4ur1j+xRHHH9tufGkLQZzGn9nv++/8fr6Ss7dI4PtV5/qP4I/+e9QXl495Pvf/gEf9yvD/wBYMz/5+/hE9j+wMB/L/wClHznefshnULh5pPGVvEf4Uj02RI19kG+vGvjJ8MG+E/iKz0wanHqv2izW6M0cJi27nddmD/ufrX3TXyZ+2ED/AMLA0b0/seL/ANHTV9DkGbY3HYx068vct2R4ee5XhMJhFOjE8Cooor9GPzcc3eul8B+Gj438ZaNof2j7J/aN1Ha+fs37N74ztrmuteg/AX5fjJ4MH/UXtf8A0YKwxEpQozlHszqw0I1K0IzPXP8AhjH/AKnWP/wXP/8AF0h/YwB6+NIz/wBw1/8A4uvpL+Civx58Q5on/F/CJ+u/2DlrWsf/AEo+b1/YxVenjOP/AMFr/wDxdch8KfDf/CI/H7UND85LwaZ/aNp9o27d2yORN9fX9fLPh3/k6rxX/wBfmq/zkr9D4IzTFZhmkIYiWzj0X8x8XxNluGwOG56ET2ivVPhH/wAi3d/9f0n/AKLjryuvVPhH/wAi3d/9f0n/AKLjr+ucP8Z+QTO3oqSCNJH2O/lp/wA9KJ4HgfY/369O6vYzI6KKKYF7QLfUJL6SfS08yeH79bN5aw+LrF9Q05PLvk/11r/fpfCe+40fWbK2fy7508xKzfC9vepr9p5MM0bo/wC/+T/lnXhVJN1J1FpyHQZNFFFe6c4UUUUAFFFFABU9rd+RvR08yB/vx1BRSaUlZgT3Fn9n+dH8yB/uSVXJqxaXnkb0dPMgf78dT/2ekf7538y0/gk/5aP/ALFY+09npMYy3gTZ50v7uBP/AB+oLiT7RJv2JH/0zjp9xcPPJ/zzT+CP+5UFNJ7sQUUUVsAUUUUAFFFFABXm2m/Ab4gftFfGh/Dx2aZodnGlw2qR/Pa2di8j+W8f9+eTY42f7A/gSvSa8Y8Ra141+AfxOT4l+Htbk3+Z5f2u6O+B4P8Anxuk/wCeH7v5P/Iex0r4viiFSeD/AHZMuS/7w+lP2m/+CetjH4OsNX+Fti/9p6Np6Wl1ozEu+qQRj/WI/a6/9D6fJXjvjj4V/G/wf+zzoPjPVbb7FbJJ9q1CNBJ/a1la/wDLB7pP/Q/40+Tf/wAtPL/Sn4Y+MNQ8eeA9D1/UvD2o+FNQ1K1jnn0bUiDNaOf+Wb+nTPIR+fnSN8oOwkhSaNldd4bqrCvxyhj8Rhfgme3PL6NT34H446x8R/DfxA+GOpf8JDHJb6zZbNkFpv8Anu38zyJI/wCDY/z/ALuT/br75/YLg+Hv/Cire48AJeFpJ3TWp9RUC7fUURPMEv8AB0dCmz5NhTFd/o37Mvwx8O+CfEvhGw8HafB4c8RSyTalYqH/AH5frzncmz/lmEI8vA2bK7PwL4L0b4d+FdO8P+HdPh0rRdOgWC2tbdSERBx3zubvu/iJNbZhmc8x5JVPiDC4OVCfPMq/FJdffwnc/wDCNvi+z8/l/wCv8n+Pyf8Ab9M/4V8uaJqVzoN7BqGlzfZLq2+45/8ARclfYXiDXrHw3pc+oajOtvaQjLyHtXyf4u15PFXiS91KGzSwS5fHkp/6Mk/6aV5JpijrF+NmtzeJ4tSl2raCPy/7OST9z1B6/wB/j7/8H/ofF+ItWv8AXNau7/U33303/ftE/gRP9is6rEciSR+RN/wCT+5SOLnnUK8cjxyRzwu9vOj+Ykkb+XIkn/PSvefBvxyt18Mzz+Ijs1C0T/lgnN9/1zT+9x0rxH7H9n+e5/1f/PP+/UFxcPcSb3oCnUnTNTxZ4sv/ABnqz3+ov/sQwp9yFP7if/HKPB95odp4ksZvENs13paP8+f9Wj/wSSR/xpWPR5lBHtPtn19r+g6N4+8M3ek6rZW+s6Fqdt5M9vOoeG5hcdD6ivzm8X/8E9/Gei/GzTNN8GXCyeBtQkaf+3b+QO+mQKRvgnTO+d+fk/56fx7NjvX0p8Nfipe+AUmspYXv9GdHkhg3/vIX9E/6Z12vw1+M8urXzWHiSeK2uLmYC1ukUpEd54gPP/fD9/y39FGtOhPnpnfP2OKUOc+P/h9/wTv8V658ZtXsvH80beCNMljlF7YPs/t6P+CBP44E/wCe39z7ke/f5kf1fN+xv8MLj4t6P47h8PxW91p8Koul2qImmvNGU8iZ4QMb40TCfw8J3RK9S0T4jaHr3iG70Wyu/Mvrbr/cm/v+W/8AHs711Um4xsU+92oqV6lSXNM1o4ajT+A+E/8AgpJpfje3/wCEO1rSvFsVj4YS7W1/shJ/s04vvndLrP8Ay3TYDH5f/LPG/wCfe+z5b/4TzW/7cj1R7pJJ0Ty/s+z9x5f/ADz8v/yJUX7QviH4hfFD4zau/wAR9Pm0DUdMd7WPRt++DT4P+WaQP/H5n+s8/wDj/wCAIiYccflx7P8A0ZX6ZwrRnChOpI8OpLnrzkjpPDfjy90PVJ7q5ebUILp991H/AMtHk/56R/7dezxyeZHv+f8A7aJXg3hu4vbfXLR9NtUvL9H+S3kT7/8An/npXvMf+r+f/WV+h4UUwooor0TMKKKKACrUE6Tp9muPufwSf3Kq0VMlzASXFu8D7H+/UccbyfcTzKvWf/Ew/wBGf/gEn9z/AOwovP8AiX77ZP8Agcn9/wD+wrL2j+DqMq29w9pMk0LvHOn3JI62H8bas0GzfDH/ANNI0/eVh0m2idCnU1mg5z5l/b0/5J/4T/7Ct1/6Ijr4fr76/bP8Ia74y8E+FrXw/ouoa5PDqE7vBpto87xp5af3Aao/Ej9nb4Q6l+zV4Wh+G1q2vfHOa10w6roGlahPfalDceQ/25HsRzHskHz8fJjt3/COKv3eZTPpMBHnonwjRXQeLfCes+B9fudF8SaRfaBrFuIzPYalbvBPDvQOm+NxvHyOh5HQiufr5U6w5peaX+IV9l/sm/s0+F7HxlrKftJeG9T8DeG204nTLrxQbrQ4ZroTR5SOR9nmP5bSfJ7ZqZTVOPMaQjzngv7OH/JTrf8A687v/wBEPX2P4J1zUY9e0bTkvZvsDzeX9k3/ALv/AJaV8/8Ah3wVD4Y/aN8Sy+H7C8PgWO91SDSNS2SPBNZfvI7V0nx8+9Nnz/x5969f0/UJ9LvoL22/dzwvvTzE8yv2Hhfnp4L3/wCb/wCRPn8Z8Z9CYormvh/rl74g0Oe6v50knS6eP92iR/u/Ljrpa/RYT5zywooorQAojkeOTej+W6UUUAXpI0v0eaJPLnT76f8AtRKo0RyPG+9H8t0rRjs/7W+eLZG6f66P+D/frn/hb7DKtvbvcP8A880T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVSqSctWAUUUVsIKKKKACiiigAooooAKKKKACiiigAooooAns7zyd6OnmQP8AfjrF8UX1ppa3Fs87/v4JPIk2PWl0ri/E14IdbeN132skEe9P+ByV+Z8e5rVyTJp4nD/b93/wI/ReBcnpZ3nUMNW+D4/d/unKXGIbWd1P7xUr638NeCNE8Mu99p+l22nvNDGjvAAHl/j2V81QeCb/AFixubqzs7zUbKNP+XGHe7/7H+xX1Y8plEan92I40Oz+58lfwfWbh8z+q+KcVGp7OlSntzc3/kp6X8MfCmr2vibT9YmtEi06Szk2SJMv8ewp8n0zVL4n+EdTfxFretG2UaYqJIZvOX7iIn8H516z4BVf+EL8Pkj/AJcIef8AgC1V+KCj/hANfIH/AC6vz+FfqFbI8L/Y3Jzy/n/7e5T+baea1/7SVRW1935cx8h+OvCej6ppOqapeaXbXWow2M3k3Ug+ddiO6V8wx/vESvrnxPDJdeG9YhhR5HaxuUSNP432PXylfaLqOkJAmoWN5p0kqfJHdQum+vyzCycbS7H9M8K1lKNWhUnv/Why1neap4bkke2nfT5H/d+Zbv8Afr2TwncTXnhnSpppHuJ3tY3kkk/jryXxAQbe3x/z0/8AZK6j4b+J9QvNVtdImmT7BDavsj8hP4I/3fz1/efBudVs3yqni8T8cv5f8R/I3GmS4bIc3qYHC/BHl+L/AAnpVFFFfph8CFFFFABRRRQAUUUUAFT2nkyb0m/d7/uP/cqCik1dASXFu8D7H+/UdWre4SdPs1x9z+CT+5Uken/ZPnu/9Wn3I/79Y+0towGW8CRp9puP9R/BH/fqC4uHuJN7/wD7FFxcPcSb3/8A2KjppPdgFFFFbAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAet/C2TzPBNuk3/AD9T7JP7n7yuX+Mkbx6ppqP/AM+U/wD6Mrovhl/yJsf/AF9T/wDoysj4pf6XqOmWr/8APlPsk/ufvP8A0Cv4f4WnbxPrf9fa/wD7cerL+CfG2nxvcQR7Ekk+TzP3dfQmh3D2+m6TPC7xzpawbJI/+udeG3n/ABJ7T+z4f7kfn3H/AD3/APsK9u0f/kC6Z/16wf8AoFf2Zh/3m550zp5PG2rSQbN8Mf8A00jT95WHSbaXpXbTo06fwIjnCiiitxBRRRQAUUUUAFFFFABRRRQAUUUUAEcjxyb0fy3Sr0kaXyPNEnlzp99P/Z0qjRHI8cm9H8t0rFxb1QBU8Fu9w3/PNE+/JJ/BVqOz/tb54tkbp/ro/wCD/fqG4uE8vyYf3cH/AKHU+05tFuMSedPL8mL5IP8A0OqtFFbJWQgoooqgCiiigAooooAKKKKACiiigAooooAKntbvyN6OnmQP9+OoKKTSkrMCe4s/s/zo/mQP9ySq5NWLS88jejp5kD/fjqf+z0j/AHzv5lr/AASf8tH/ANisefk0mMZbwJs86X93An/j9QXEn2iTfsSP/pnHT7i4eeT/AJ5p/BH/AHKgppPdiCiiitgCiiigDgfjpZ+Jbr4a3kPhCeaDXvtUEkMkFykL7N/z/O5H86+Wx4d/aNVSDq2pE9iNftv/AI9X2Z4i/wCQTJXG1/Jnibmk8HncKapwn7kfij/ekfsHCeWwxWA5+eUPe+zL/CfHninxd8aPB+vWei6r4i1a01G8jR7eFNUWQOjPtT50fZ99P0rq5PDP7R8buj6vqIkT5CP7etv/AI9UP7SzY+NXhAZx/odt/wClU1fUmof8hK7/AOu8n/oyvgcfmKwtChUp0IP2n9097BZe8RXr0515+5/ePl4eHv2jsHOq6j/4P7b/AOPV4R4zt9XtfFmsw+IJJJdaW6kS8klfzHMwfD5bPzc/zr9Eq+BPjVn/AIW34zz/ANBq8/8AR716nDuYTxtSpF04Qt/KeZn+AhhKMJqpOf8AiOFooor7U+EJVJZgBye1fSNvoX7RH2WDydS1HyPJj8v/AIntt/q/L+T/AJa/3K+cIcrKhA5r9JtP/wCQTpv/AF5Qf+iI6+S4ixs8HGm1CM73+L5H2PD2Dhi5TvUnD/CfNGqzfGrwho9xqPiXXdW07TfMjjN5Dq8M3lO/3N6JIfkqroEfx/8AEmmxajp2uatc2EjvGk7a1CgbZ1+/JXq37Tn/ACRXVv8Ar9tf/Q5K1/2ZbhLj4GeH7W6/dp5915M/9z9//wCgV4P9pL+zvrfsIc/Py25T2/7Pf1/6r7efJyc3xf3jx268K/tD3EEkE2p6hLDPG6Oj6/bfOj9f+W1Q+Bvg38Y/BNreXNiLnSbW62b4LHWrWN7t0Pyf8tOdnmZ/GvrWPS/7P8ye/T92n3I/+e//ANhVG8vHvJ97/wDAI/7leU+I68IOn7GGv93/AO2O/wDsGjUnz+2n/wCBHzRc6X+0jqE0ksmqakPZNatkVP8AyJiuP03xd8aNW8a3PhO08R6s+v20kySW39qKgR4f9YA+/Zx5fr2r7Er5g+H2f+GyvEHPH9oax/6Lnr18BmMcXTrTnQguSHN8J5+Ny94WdGEK0/fly/ES/wDCO/tGk/8AIX1ID/a161/pNXlnxe03xxp+vWa+PLq4utVazVoZLi7S6cwb32nzFdv49/Ga+7a+Sv2wgP8AhYOjnPzf2NFxj/ptNW+Q5pPG4t03RhDT7MTPOsshhMJz+0nP/FI+f6KKK/Qj84HHqa6LwNb6zeeMNIh8PSSxa291GtlJDLscTE/Jhu3Nc53r0L4C8/GTwb/2F7X/ANGCsMRLkozl2R1YaHtK0IHq3/CN/tF5/wCQxqP/AIPrb/49XN+N/FHxq+HP2L+3/EerWS3m/wAkx6ok+/Z9/wC47/3/ANa+u6+dP2yf+Pfwdk4/4/P/AGjX5zlOZrH4yOHqUIJa/ZP0bMcseEwcq8K0/wDwIrQ6H+0TcQRsuq6oY3jV0P8Abtt9x+n/AC070nw/+FPjzw58QW8T+KbLMUwuPtV3JqEM8jTPG+Cdkm/Jf0r6S0//AJBOmf8AXla/+iI6oeJP+QO/++lenwzn1Z51hqEKcIc04/Z/vGOaZHRhl1WvOpOfJCX2jk69U+Ef/It3f/X9J/6LjryuvVPhH/yLd3/1/Sf+i46/s7D/ABn4TM7erUE6Tp9muPufwSf3Kq0V6UlzIzJLi3eB9j/fqOON5PuJ5lXrP/iYf6M//AJP7n/2FF5/xL99sn/A5P7/AP8AYVj7T7HUZVt7h7SZJoXeOdPuSR1sP421dodm+GP/AKaRp+8rCo2+9OpQp1NZoOcWiiitxBRRRQAUUUUAFFFWoIE2edL+7gT/AMfqW7ALb26eX5037uBP/H6f/ajyP5cyeZa/c8j+5UFxcvcSf880/gj/ALlQVlyc2sxk9xafZ/nR/Mgf7klQVPaXnkb0dPMgf78dFxafZ/nR/Mgf7klNNrRiIKKKK2AKKKKACiiilcBAc1w3xgk8UWfhyO68N3T28drPHd3Ulp/x9psk8yOSP/rm8fmf9s/+uldleXkGn2M91eTJb2kKeZPcSf6tI65uT4n+HrfwlH4lS+8y0d/LS3j+Sd5/+eez+B68TM3QqUJ0Zz5TOZ9SfsX/ALXknx80+fwt4hhFv470q2F1NNZx/wCi6jah0Q3Sdo33uA8f/fGeifV4y2a+C/8Agmr4z1LXL7x9aReCLHTdCku/t8niDTY9kf2o7ALF9/3yiHeNn3Od4TzI9/3PqMC3Wn3MLTvaI8TKZkba6cfeB/Wv56rwUKjUD6XBznOjCczyn4ufFl9PaXQNAZ5NSfMdzdwHPk/9M4z3m/8AQKy/FH/CwND+Heh3c+qJbzWL774RyHziPMHkb36SY/j/AL/+3mm/C3Q/CHh211jxDJrsOrnTZ5ES5kR0EEf8D+X/AMtJJP8Anp/H/BXP3V1rXxy8Vi0sVlsNGtZd/wAwylqn/PST+/M//PPoOn993xInObCOfxF8b/EkED7YLGzKecIzmG26neP77v8AoP8Ax+5Z/AXXpb7VbeWWGCC2T/Rbr+C5fqnHVE/vj/0OvdfC/hfTvCejwabp0IjghHfBd2wMu5xyx9a8f+L3xcNw03h/Q7rYm/yL2/jfvz+5T3P9+kKdKEIc9Y8huLd7OeeGby454Xkjfy38z95UlvbpHH583+r/AII/79FvbpHHvf8Adx/wR/36juLh7iTe9B5+pP8AbPtnyXP7v+5J/cqCSN7eTY9R1YjkSSPyJv8AgEn9ygNSvXq/wbj8L63puo+HNRs0/tO8/wBZJM/N0if6vZ/cdP7nXjf3ryySN7eTY9RxyeXJvR/LkR96SRv5ciSUF058kz1W1+AGpN4gvra5vFi0eNd9tfKQ88zgny02f7AHz/hsx/yz5bw/FD8O/iFBF4r08SJa/vPMX540/uXSf30/z9+OvWPhT8Wl8ULHperypFrUfCv9yO77ZHo+P+Wf/wBcL1njnwLp3j7STa3f7qeM77e6QHzIX9R/8TTO32MJw56J5h8SPhtCL7TvEPhvU7LTPt11DseS5EcfnPjy54XzyefuDh69tsYp47GFJ5/tE6pteby9m9v72yvkXxHo+o6DcDRNYLkWfmPBBv3wbH6un+w9e7fBPVfEOpeG3/tZC9ih22N5M/7+ePj25Xp8/wDP79BpRn79j8uP2i/BPxA+CPxe1RviFqF14jm1mZ7qDxFJ/q9Xj/2E/wCWMifu08hPufJs+TZXM+J5dQ8G+KrjRdd0i80S7hRHngvoNk6b/nSTZ/cr9oNd8N6R4gWx/tfTbPUjYXKX1sbuBJjbzoTsmjL52SJ2ccjmvKv2mf2X9B/aP8KJDcSJo/iexTOma7FCHeA5OY3TP7yF+6GvosFnuJwcPYw2PPngJx53RmfmVo/7zWNNgh1SHS57qfy7W7kfy/3le4x6xY/8JH/wj321JNdSDz/sn/LTZ/uf+069r8Tf8E0fAGr2PhO103VtS0hNLAh1aVP3kurwZLuT/wA8Ji//AC0jGAjFNuPL2X9U/YJ+F+kfFS++Id9qElp4PtbaS7uvD1xLstYZxGfMuXuC+8Q7BvKH+P59+z5K+ohxfyfDAy+q4m1zwLR/EGl+IJL9NNvYLySyn8i68h/uSVo/Wvl5tVsPC/xKvZ/hneX8+gJcummPqqbHubX/AJ5zp/zz/wDH9nlv+7evovw74jtfFNj59t+7kT/X28n+shevt8nzZ5lC84chwwnz6GvRR/7I/l0V9MUFSW9u9xJsT79FvbvcSbE+/U1xcJAn2a3+5/HJ/frJt7IBbi4SCP7Lbf6v+OT+/Rb3CTx/Zbn/AFf8En9yqlFL2aAkuLd7eTY/36jq1b3CTp9muPufwSf3KhuIHt32P9+mm9mBHH96vm79ju4j0r/goF4gvNSmj0+wkn8QJ9ou5BBH8wn/AI3+SvpBTuGa474yfD2X4x/D9/CkmryaZE93Bdmd4fO2bPM/g3p/z0r5DiXJ5ZthuWn8cT0cHW9jPmZ6T8Yv2Hfg9+0Z8XNZ8Yap8RNQTXtdeD/QdG1PT3T93BHH8ifO7/JHXM/8OivhH/0OHjU/jZf/ABuvnTwn8EYv2RtdsvjCNbXxf/wiM8d7/Yr2f2L7VvkEGzz98mw/v9/3H+5XqCf8Fj4I49n/AApr/wAuv/7kr8AzDKc0yucaMj63DYjDV/fmddrH/BKz4HaI8a3/AMRfEmnTzJ5kcd9fafDv/wC/iJWV/wAFXNX03WPhR8PbfSNYsdTeHV5k2WV7DO/l/ZY05Ebn+5XhPxy+MkX7fOoaVqn9jjwCvheD7D5f2r+05LsTu77/APVw7Anl/wDj9c98LfhT/wAKx1a/voda+3vdWv2T/j18jZ+8jk/56f8ATOveynhrMMc6Nev8H/bpw1sZRp88IHS+CuPBPhz/ALBdr/6AlbFH+s+/VvQ9PTWNcsNO8/7P9qfy/M+/sr90pU/ZU/ZnzDd3c9N+E/8AyLE//X9J/wCi467Gsfwf4b/4RfSpLL7V9s3z+f5mzZ/yzj/+IrZFe3R92Bzi0UUVsIKKKnghed/+eaJ9+ST+Ck3bcAt7d7h/+eaJ9+ST+CnyXfl7Et98aJ8/mf8ALR3/AL9LcXCeX5MP7uD/ANDqpWNnLVjL0kaX6PNEnlzp99P/AGolUaI5Hjk3o/lulXpI0v45Jok8udPvx/8AtRKP4XoBRooorcQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBQ1u8fTdJurqIJJJCnmJ5lcZcTf2xcf2hf7I4I08v93/H/ALldv4gt0/4RnUp7n/UJH/33Xmt1dvdPvb93GnyJHH9xK/lLxax+Jp4yGC9p+55Obl/vc0j+pPCDL8PPB1sbyfvufk5v7vLE9s/Z41R7hvE2+P8AcbrZEgj+4n+ur1drcqokX50b7j145+zp/q/E31tv5zV7JZ3GxWR/nRvvx1/M9WftJrnPqc9h7PMa/J/d/wDSUfTPgL/kSfDv/XhD/wCgLVP4pf8AJO9e/wCvZ6ueA9v/AAhegbPufYYcf98VT+KX/JO9e/69nr99rf8AIqn/AIP/AG0/nWj/AMjCH+P/ANuPm2T/AFZrxb9or/kKeGv+va6/9DSvapP9Wa8V/aK/5Cnhn/r2uv8A0NK/n/D/ABs/pTIP+RjT/wC3v/SWeNXenxX0ccTl/wB3/wA86l+G+yz8abHdI0RJ4/Mkf/pnUm3kmuTvI/Murvf/AM9K/p3wmzDE1K1bDTn7kIfD/wBvHxXi/l+Gp0KOMhT9+c/el/26fQ8ciSJ5iOkif9M6K8y+H/jT7P8A2L4eTTv3bv5f2jz/AO/JJJ/q/Lr02v6mpz5z+XQooorUAooooAKKKKACiirUEKRp9puP9R/An9+pbsAtvAkafabj/UfwR/36f9r+3/Jcfu/7kn9z/wCwqrcXD3D73/8A2KjrL2d9WBJcW728mx/v1HVq3uEnT7Ncfc/gk/uVDcW728mx/v1Sb2YEdFFFagFFFFABRRRQAUUUUAFFFFABRRRQAUUVJDbvO+xPv0bbgeqfC6N5PCNuif8AP1P/AOjKwfjDcfZ7qwgh/wBW9lPvk/v/ALyuh+HMiW/g2KCF/MT7VPvk/v8A7yuX+MX/ACEtM/68p/8A0ZX8P8K8j8T63/X2v/7cerL+CfJml3iXmmwWV5J+78v9xP8A88P/ALCvedPt3s9NsY3++lrB/wCi6+d7f/j1T/cr6P8AD9wlxoem21x9z7LBsk/ufu6/sfD/ALuZ50woqS4t3gk2P9+o69i9zMKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFT29u9w/8AzzRPvySfwUW9u9w//PNE+/JJ/BUlxcJ5fkw/u4P/AEOsW3sgEkvPL2Jb740T5/M/5aO/9+pJI0v0eaJPLnT76f8AtRKo0RyPHJvR/LdKXsuwwoq7JGl+jzRJ5c6ffT/2dKpVrGXMIKKKKoAooooAKKKKACiiigAooooAKKKKACiirVvAm3zpf3cCf+P1LfKBJp+nvcxyXT2t1JaJ8n7iB33v/cqTdeySbJtLvZLT7nkfYZ/k/wDIddH4V/aOm+GGmz6Qvh9tSSaf7Wkiah5Oz93HH5fl7H/uVpXH7bU9vazzf8IZJ8ieZ/yHf/tNfg+c+JMMrzCtgqlG7h/e/wDtT7rB8MYrFUYV4L4zgtQ0+ez8uR4Z44H+5JPA8dVDX0Z+0VeZ8HaEZv3iTaim+Pf/ANOklfPdxZ+R86P5kD/ckr9O4dzmedYKGJnDkufLYzC/Va3IQUUUV9YecOjt5rr/AFNrPceX/wA8IHf/ANAqX+z73/oHah/4BT//ABuvbP2XXdv+EmCP5Yaa1T/0ZXPXH7Z89vdTwf8ACHySeRO8H/IZ/uSeX/zxr8W4h8Qf7Ax88HUo/D/e/u838p9dluQ1syo81A+dfjlqviDwb8Nb7VtG0Ke/1FLmGBIJ9OunjdHf5/uV8x/8L4+K/wD0TaP/AMFV/wD/AByv0iT9tadPu+D5P/B4f/jNP/4bcuR/zKU//g8/+0V+LZ7xVlmfYn61iaPv/D9o+3wGT5vltH2NGX/pJ+TfxC1Lx/488Uaf4hv/AARdWs2mQpGiW2l3KQbEd5Pn35/vnvXUXH7WHjeV2m/4RXRiXfzDi2uv/j9fsN8P/i5dfFr4a+INZaym0cwG8tPI+2eePktt+/fsT+/Xw3p+qXv9m2n+mzf6hP43rwcwzTB06cL0OeHTyPQy3L8ZVqVv33JM+VZv2tPGNqczeGNDSP1kt7of+168g8TX2peMfEmp65Lpzxz6pdPdOltC2zLvvOz25r6c/bFuZpvhzoJmmklP9rPjzGz/AMsRXpfwp1C6T4VeD0juZoo/7Jh6Px1rohmtDA4GGPoUEuf3bXIlldfHYqeBr1vg94+Brfw3q19dJbW+mXk9xM2xI44XZ3P0rR/4Vj4u/wChW1v/AMF03/xFfqL8FtQupPjF4IR7qYx/2knV3/55yV9KfFr9pG6+E/ia10hdFuNW86zS+M/9p+Rs+d02bNj/ANyuvDcSQq0fbVociPKxXD86WI9hRnzs/Cpfhr4uVgx8Ka0x/wCwfN/8RXu8Px4+K1tDBAnw1j8uFEgT/iU3v8CbP79fpV/w21cf9Chcf+D3/wC0Un/DbFz/ANCjP/4PP/tNYYnPMsxSSrQ5/wDwI68PlWa4XWiuX/wE/Ln4gfEP4lfETwnPoV34AmsrWeRHL2umXu/5PeR3p/gT4t+MPhT4Tt9EvfC1slnZyO8MN/ZXSXUzP8/aRPkr9dfhT+0pefErxXb6VJoUukRzQzTRz/2t5+/Ym/7mxK8L/aa1K5T4260PtMw/0azPEzj/AJYmufEZngIYL93R5ocx04TB5hVxtq9XknyHw1D+194x1STydQ8L6NBF1gnSzutkP/kb7lZU37VXjW3uJIW8L6LJKn/POK6f/wBrV9P63qN1J4d1lHuppP8AiXXX8f8A0wkrwv8AYn1iWTTfGdnLcvEiSWXkzRv/AKv/AF/X/YrmoYzBYjC18X9VXucvU7q2HxtDE0cL9Z+Pm+yciv7Vvjctz4V0YL6/Zrr/AOP1zPh3V/HVn8Rp/iDY+DLq8ub6S5uQo0+5e1JnD52bOcfOcfNX21cXGo2c/kTXU3mf9d6+xNP+Jd18N/2bPC3iLyLjVDBpenRi1N48O/f5aff+f1rTLs0w1RTSw3Iuvmc+ZZfjKPJ++53zH5R/8L4+Kv8A0TZP/BTf/wDxyvNfidJ48+K2uWmp6h4IvrOeC2W2SOy0268vZvdx9/ef4zX63f8ADbVx/wBCjcf+D3/7RSf8NsXH/QoXH/g8/wDtNb0c3yrCz56ELfJnLXy/N8VDkrK//gJ+Lh+GPi7t4W1r/wAF03/xFQX3gfxJpcKTX2g6lYxs+xHuLN4xu9OV/Sv2kuv24rmC1nn/AOESuMInmf8AId/+0V0f7XmoXP8AwrjQZPtM0e/VoX/1z/8APrPXq/6x06lGdSiubkPMhkNf20Kdf3Oc/Cb+w9QXrYXHPT901bXg/Vr7wb4t0rWbe1V9S067juooZkb5mQ7xx+Ffof8A2pe/8/c//fb18w+LrgyftqaFI8jeadR0jL7uf9Rb1z5fxAsz9pCVG1oyZ34zIPqHJOFb7USjB+1R42mKIPCOjOzfIn+iXWf/AEdWB8VNa+IPxX/ssan4Gu7AaeZdn2HS7pN2/Znfv3/3B+tfa2h6pex65ov+mzf8hG1/5bv/AM946+0/jh8dLj4O3OlRrpkurrqjXR5v/J8nyfL/ANh/79eXgM0wMVPEqgoch15jg8bFwwvtufnPyNg+O3xVtra3hT4bRbIUjgQPpN6ThE8v+/W94X+J3j3xnrkel654JbRdMkR5Hmh029Em+NC6cu71+hX/AA21c/8AQo3H/g8/+00n/DbFyOnhGcf9xz/7TXdgc6yjB4mGKhTXuS5vtHPWy/Oq1GdCcvj/AMJ8T/Z7n/oHah/4Az//ABuvV/hHZ3R8OX3+g3v/AB/P/wAus/8Azzj/AOmde/8A/Dbl1/0J8n/g9P8A8Zr1P4Y/Fuf4t+BtW1htPm0gwTTWPkC98/7kCP5m/Yn9+v1bCeKFOtVVKjR/8m/+1PkK/DGJw8OeufLjVLb27zvsT79V9Ht3uLW0RPv+SlXridII/stt9z+OT+/X9Ec3Mkl1PhHvYLi4SCL7Lbfc/jk/v1JZxz3lv5D2t1In8E8dq8myqA7V9F/D3xVJ4L/Z3m10QveJpttfXX2Xz9nnbJ5Pk3/wV8ZxJnX9gYL61yc/vf5/5HpYDB/Xa3sT5/uNJvbeSRH069/8BZP/AI3Uf9nXv/QO1D/wCn/+Ir1C3/bT8yTZN4SljT/np/bX/wBoplx+2Xc2c7pN4Ll8z/sO/wD2ivyn/iLVL/nx/wCTS/8AkT6//U7Fdv6/8CPLKKKK/o4/PAooooAKKKtQQJs86X93An/j9S3YBbeBNnnS/u4E/wDH6juLh55P+eafwR/3KLi4eeT/AJ5p/BH/AHKgrNJ7sAooorYAqe1u/I3o6eZA/wB+OoKKTSkrMCe4tPs/zo/mQP8AckqCp7S88jejp5kD/fjouLT7P86P5kD/AHJKyTadmBBRRRWoHm3xX+K6eB4JNO02dP7e2eY8kn+rsk/56P8A7f8A0z/4H/v9Bpf7Ovxu1L9mmTX4FmkuJbjzoNCkjdNZl08p8zpJ5nDmT5xAfn2ceZnEdeQ6ro+ofA/4kaV4n+xW3iuxh1D7dAdZTz4LqT+5P/00/j8z+/8AP/yz2V+sHwF+OPhv49eCLbX/AA9P5bpiK906c/v7KfqY3H54fo/6V+G5zmeYRxM1L3DXBUYYqc+c+DviT+yD8dda03wBpM/2DWpNS8tNQ+ytsj026/eP5l6//LREj/5aR/x/u9nzo7/Rlt/wTx+G0/wgtvCE/wBo/t1ZvtU3i6BUj1F7jpJyQ6eVtzH5OCijkfP89fWnNeSfFf4vjw6z6RobxyawP9dO6b47b6/7ftXy1bF168+epM9f6nhqHvzOt8C+BtG+GvhnTvDvhzTYNK0WwjEFtbW4+Vffnln6szvl3fJJOa19e1HT9L0O7uNXeFNPSP8AfGddybcfdP8Ae/8Ar14lqH7QWqSXWlvZ2K28EI3X8L/8tXx9xHPCJ71yfxF+IN74+1Ul/Mg0q2kza2v/ALO/+3/6BXIaTxMOT3ClofhqHxt4vGmaKn9k2Uzu8RuWLvDCnt/G/wD0zzxX1F4W8L6d4T0eDTdPh8qCPkk4Lu2Bl3OPvV8veBNP0zUPFmnw6pqE2mQeckiTwjY/n/wR7/4P9/8A+OV6T8Xviuytc6Boly6yL8l7fxt/q/8AphH/ALf99/8Aln9c7AwozhCHOR/FL4pXGoXsnhvw3JJPcSP9nmubQ/Oz5/1EHv8A35O3/oHT/DT4P2fhjSLiTVreG71K8g+zzxn544Yf+eCe1Yv7P3h3Q102TU4ZEuNbxskjePZ9lT/nmnt/t967X4jfEez8B6aMAXWpzj/RrTP3uo3vjog9aDoj/wA/pnF/EnTvCvgPwPLo62f2u6v5N9tFI/74P/BJv6oifl/B/Hz4bXqPgHwDf/EbUn8TeJHe4spm3xhxs+2fh/BD6Dv/AOh8H4qs9IsPEV9DoV69/pqP8k8n/otH/jT/AKaf/t0HHW/nMqiiikc+pf0+4g8yBLyD7RAjx/u9/l74/wDnn5lfQulaX4P+IH9kavBYI0mk4ijtjHs8n+4jx/7HDp6dU61458J9M8O6v4ojtdfbPmJ5drbv/qJpP+mj/wB/+5HXQa1ourfBTxEmoadNJcaNNJ5aTSf+iJ//AGST/Ls7KHuQ5zovi18I3vJ28R+HUeLUkkM1zaW/yefj/lpH6Tf+h1f+FPxcTxOqaTrUiRax/wAsZ/uC5H0/gf8A2K7PQ/GEPi7w699pOxrpVKG3uGKmObH3Hr5b12TU/wDhIL19ZR4NY87fcmM7Nj/9M9n/AGzoN5zVGXPA9l/aEv7ePR9Ntp9JkuJ5py0GokYjtufubz/G/TZ+P8FcH8MPidN4LuxZ3byT6JM/zx5LvbP/AH0/9nTt/wCjOx8IfE/SvFXh+40Pxk8AkjhIknnGI7lF6k/3H9v++K8bvPsv2qf7A80ljvk8iS7/ANZs/wCWe+g560/f54HU/ED4j6h421hJLd5rDTrObfZpG+x9/wDz2f8A2/8A0XXtPwg8bX/jXw3JJqFq0dxayCB7qNP3F1z9+P8A9m9DXjXw3+G9z48vneYtb6HbPsmnT78z/wBxP/jlfTen6fa6PYwWdpAlvawpsSNOERfSg6MNzzlzsraxrFjoWmXWo6jcw2VjbQvNNc3MnlpEiDLuznhVAGa/LH9rj9rPUf2jtZfwt4X86z+HlrNvjjf5H1p06zz/ANyBP4I/+Bv8/lont/8AwUvj+I1xp/h61sWhHwvuXS3vY7Sby55dR8zzIxdA/fgxGNgj/j37/wDlnXxrpelppcGxP3kj/fk/v19PkuUzx0+f7B5+NrVJz9jD4Q0vS00uDYn7yR/vyf361tL1S60e6+1WE728+zy/MjqpUlvbvcTxwQo9xO7+Wkcf+sev1yhThh4clM44Q9n7hs+C/El74f1WNLOB9Qjun8uSw/5aTyf89E/269ygt3uJtiffrmvAfgT/AIR+Pe+y41aZPnk/5Zwp/wA84/8A45XWXFwkEf2W3/1f8cn9+vVoxnHQYk86QJ9mt/ufxyf36q0UV2pWMgoooqgCrUE6Tp9muPufwSf3Kq0VLXMBJcW7277H+/UdWre4SdPs1x9z+CT+5UNxbvbybH+/UJu9mBXuLeC8t3guYYbyB/v29wiSRv8A8AevPfGHwnTXNY+1WGl+H7eDyETy5LFI/n/ef3I69GpMVlUw9Op8aGeD654PfwXPBA8GnxyXSeZ/oEHl/c/v/u0rNr33VPD+naxJG9/ZQ3kifckuE+5Wf/wg/h7/AKAll/3xWH1ZL4C+c4LT/hnqmoWMF1DdWXlzJHOnmO/8f/bOvQfCfh/+w9DtLW5gtZLuHzP38af9NP79a1vbpZwRwQokcCJ5aRx/wJUldEKMYGYUUUV0gFFFTwQvO/8AzzRPvySfwUm7bgFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6Li4Ty/Jh/dwf+h1UrJJvVjCiiithBRHI8cm9H8t0oooAvSRpfo80SeXOn30/wDaiVRojkeN96P5bpV6SNL+OSaJPLnT78f/ALUSsP4QyjRRRW4gooooAKKKKACiiigAooooAKt28CbPOm/dwJ/4/RbwJs86X93An/j9R3Fw88n/ADzT+CP+5WDblohlDxFHPrmn3lqmyOSRPLSP/lmleb6ppc+j3XkTOkkmzzP3den+p71wXjT/AJDWT/zwj/8ARklfgfitkuGnl/8Aaf24csT9/wDCPN8TTzCeVw/gz5pf9ve6bnwy+I1n8PYNX+2Wd3d/afJdfs2z+Df/AH3/ANuvo4/Kqn++m+vjSZfNtZ0j++6V9ReEfiFovjSeW00+4llntrZHl32zp8n3K/j6tC65+x+28T5fy1PrVOG/xf8Akp9LfDX4gWWoQ6V4fht7pbm2s0RnkVAnyIn+371X+J/jy1Wx1nw81rdC4eHy/P2J5PzJu6764v4PH/i4Vt/17Tn/ANAqL4mz7fiJrMbfPG5hLp/2xSvvp51iv7D9p/e5P+3eU/BoZXQWbez6KPP/AOTHFalfJY6Zd3zo7x2sLzOif7Cb6+cvif4+s/H1zpU9pb3VpHZwzRt9p2fPvdP7j/7Fey/ErxppPhmx1DSbiS4W5vLGXyjHbu6NvR0T56+aI/3caGvhaEeX94fvnDGB5pfWqkHp8BBeagmnxxu6P8//ADzrm5JPMnuH/vv5laviD/Uwf9dP/ab1jV/XvhhlOGo5b/aMfjnzf+lH4N4rZtia+bf2XP4Icsv/ACU2vBf/ACOOg/8AX1XuNeHeC/8AkcdB/wCvqvca/dsMfhkwooortMwooooAKKKtQQpGn2m4/wBR/An9+pbsAtvAkafabj/UfwR/36guLh7iTe//AOxRcXD3Em9//wBio6zSd7sAooorYAq1BOk6fZrj7n8En9yqtFS1zASXFu9vJsf79R1at7hJ0+zXH3P4JP7lQ3Fu9vJsf79Qm9mBHRRRWoBRRRQAUUUUAFFFFABRRUlvbvPJsT79LYAt7d55Niffqe4uEgj+y23+r/jk/v0XE6QJ9ltv9X/HJ/fqoayX7x3Yz1X4Zf8AImx/9fU//oyud+MX/IS03/rxn/8ARldF8Mv+RNj/AOvqf/0ZXO/GL/kJab/14z/+jK/iPhX/AJOhW/6+1/8A249Of8A+Pbf/AI9U/wByvoTR/wDkC6Z/16wf+i6+e7f/AI9U/wByvoTR/wDkC6b/ANesH/ouv7Ow/wAZ58zWt7hJ4/stz/q/4JP7lQXFu8Emx/v1HVu3nSdPstz/AKv+CT+5Xd/D1RBUoqS4t3gk2P8AfqOtU7iCiiimAUUUUAFFFFABRRRQAVPb273D/wDPNE+/JJ/BRb273D/880T78kn8FSXFwnl+TD+7g/8AQ6xbeyASedPL8mL5IP8A0OqtFFapcoBRRRTAI5Hjk3o/lulXpI0vkeaJPLnT76f+zpVGiOR45N6P5bpWLi3qAUVdkjS/R5ok8udPvp/7OlUq0jLmAKKKKoAooooAKKKKACiiigAooq1BAmzzpf3cCf8Aj9S3YBbeBNnnS/u4E/8AH6juLh55P+eafwR/3KLi4eeT/nmn8Ef9yoKzSd7sDj/Fn/IV/wC2CVzeqf8AINu/+uD10niz/kK/9sErm9U/5Bt3/wBcHr+CeNP+Sjxn+M/pfh7/AJFdH/AfX37XHii18J/D/wAJvcwTXHnaikafZ9n/AD6SV8y2fxn0u33o+nahJA/34/k/+OV7l+3h/wAk18Cf9hj/ANsJK+Mq/qfgiXLktP8A7e/9KPwXNf8Aej6D0+8g1jSoNQs9/wBkm8zZJJ/sSVOPSsP4V3n2fwVYo6eZA7z74/8AtvJXSXFp9n+dH8yB/uSV+l06nu++eGe4fsr/AOu8R/8AXay/9qV8j6h/yFNS/wCv2f8A9HyV9cfsr/6/xH/12sv/AGpXyPqH/IU1L/r9n/8AR8lfxp4nf8jut/X2Yn7bwR/u8/6+1Igooor8gP0o+nP2dbr+zf2f/G90UZ4oJ9Sk2J/0z0+PP6Cvzxh/ay8G2trBG+na8fLRE4hh7f8AA6/Rf9mqIXXwL8YwSxpJbPdX8bpJ/Gn2KCviix+E/gaaxgeTwho5/coeYZP/AI5X1qqYCnhaKxdOUz42jDG1MXXeFnGB8+/Hf45aD8UPDOm6ZpFlqUc1tevdSPfJGiFDHswNjmuw8G/tO+FPDngvQNJutP1l7mws1tZmt0hKM3+x89es/wDCpfAn/Ql6N/35k/8AjlMPwn8DHr4M0c/9sZP/AI5W08zyqpQjhHQnyQNIZdmdOtPFe2hzzLf7M/7R3hnxp8f/AIe6TY6frCXV1rMMaNcJDs/j/wBv3r6O/a5/5KZpP/YCj/8AR01eTfAH4c+EtH+OHgS50/wxp1ldQ6sjxz26yb0/dyf9NK9Y/a8P/FzNJ/7Asf8A6OmrkxlTCVMH/scORDo08TTzKH1qfO+X/M8Vq1Z2aRwfarz/AFH/ACzj/wCe9FnZpHB9qvP9R/yzj/571BeXj3k+9/8AgEf9yvlz6rU9Z/Zju3vfjlYO/wD0DL3ZH/c+SvJv20v2gvDnw7/aN8T6NqWn6tPdW9vp/wA9okJTm1ST+/8A9NK9R/ZXOPjbpn/Xhef+gCub/as8A+F/EHx88RXmq+HdN1C6eCw3zXCvvf8A0ZB/QflX1GDqYSngf9rhzrmPlcVDEvM/9lnyPkPlq+/aw8HX2lX9rHpuub7m1ngXzIIeHkjkTP3/APppXl/7P/xi0L4Wx+I49Ws9QmbUDbeT9hRDt8vzN+/e3+3/APqr6WHwm8Cr08GaOP8AtjJ/8co/4VN4F/6EzR/+/Mn/AMcrrp5nlVKhOhChPknuKeXZnUrQrzrQ54HH2f7YHgi+jSyu9M1xE/5Zz+TB+59h8/3K++fGGoLq/wCxB4Q1CFHS2utL0SdI3/1mx5I8V8df8Kn8CA5HgzRv+/Mn/wAcr7P8X2lpb/sW+F7FUjs7OHS9ESHy/uJ+8jp06mAqUZ/VISh7plXhjadah9anGfvny3RT7i3eznkSZP3lMr4w+zKeof8AINu/+uElfTH7fnjnT/hv8E/DesapDdXFq+vW1qUtNm/f9inP8f8AuV8z6h/yDbv/AK4SV9WftraJpniX4S+GrHVrC21K2TW7aXyLpfk3/Yp6+jyn2apVvrC/dnzWbe19vQ9j8Z+eS/tceC2z/wAS/XR/2xg/+Lrx7Wvizomp/tCWPjmK2vxpEN1ZTtDIieeRDHGjjG7bz5f619Qf8Kk8DY/5EzR/+/Un/wAco/4VL4FHI8GaPn/rjJ/8cr0MLmeU4PmdGjNc/unPicuzTFuEKlaHue8cr4O/ak8J6z4s0Czh0zXBPLqlqieZHD5f+vT/AG6+7f22ht1bwgOwOpj/ANJa+TvD/wAMfBVn4i0eaHwppUc66ja7Xjhk+T9+n/TSvrH9ths6p4QPvqf/ALa1hWqYGpg5/VIcgoU8TTzHD/Wp8/xHzVRRRXyR9cFfVf7K0nl/BrxPJ/cv71//ACVhr5Xt7d7yfZCn7yvqf9ncQR/A3xiltJ5kf2jUN8n9/wD0FK+gyGH+3Uz5viD/AHOX9dD5qt/jBolvp1va21rqEieQm+42J8//AJEq94c8cWXii+ktba1ureRIPP8AMuNn/wAXXh2l/wDIJsP+uCf+i69C+Ef/ACMl3/14yf8AoyOv9AKMp3SP5xd7nqZ6V7dZ/wDJoviP/sF6j/6PkrxFq9us/wDk0XxH/wBgvUf/AEfJX5t4mf8AImh/i/8AbZH0PDX+/r+vtRPkqr1vcJcQfZbx/wB3/wAsLj+5/wDYVRor+KdT+kT0iiiiv9Mz+Rwooq1BAmzzpf3cCf8Aj9S3YBbeBNnnS/u4E/8AH6juLh55P+eafwR/3KLi4eeT/nmn8Ef9yoKzSe7AKKKK2AKKKKACiiigAqe1u/I3o6eZA/346gopNKSswJ7i0+z/ADo/mQP9ySoKntLzyN6OnmQP9+Oi4tPs/wA6P5kD/ckrJNp2YGbqml2ur6bcadfwpeWl0mx7eT+OvGNMvvFv7K3xFsPFXhS9kls3Pk+ZP/qL2D/n0u//AGR/+Bp/cr3H71Q6hp9rq9jPZX9ql5aXSeXPbyfx14WbZTTzGn/eIlFx1pn218A/j94b/aC8Exa5ocrQXMJW31DS7iQfaLCfvHIP/QX6OOfpQ+LHwhGqefr3h+H/AImJ/eXVpGMfaf8AbT/pp/6HX5teD9H+I/wH+Mmi6h8NLe812+1Cf7LaWkab49QT78lpdf8AkR9/8Gzf8mySv1y0OW+utLt5r+2js714kea2hm85IXIBZEfYm8dt2B9K/B8bhZ4St7GZ7+GrfWqfJM+O6K9U+PWj6Fp2uQXNlOINaucvdWkafu2T/ns/9x//AEPFeV1xHHUh7OfIFWLe3SOPz5v9X/BH/fot7dI4/Pm/1f8ABH/fqO4uHuJN70EanpXgn4paR4P8J6j5el7df8z5OcJcj+D5/wCBE6bO3/A6g+Hfhc/FLxPe6t4ivEvPJdHmtd/zzf8APP8Ad/wQ/wCH+/XnFSW9xPZvI8M728jpJA/lv5e+N/8AWR0zT2n856r8SPiTdeML5PCnhNJJ4JpPJeS1x/pX+wn/AEw/vv8A+yff82+P0Oj/ALO/gfT9c1zV0u9SucwJo8R2PeTcYjh/2EBy7v8A7/ohxdS+N1r+z7Yy+JblllST/RY9OU4lv5P+eEf9z+/v/grwP4U/C/xz+3h8XtS8WeLb64tPDdrN5Oo6jZjZHBHkP/Ztj+fzv/Bv3vl3q4Q+3IiVb2nuQ+MtW/7SGtSfA+/8XP4Rm+12uo/2U+pRwP8A2Kjv8/mSP5m//pn5f9/y/wB5+82Va8bftHal4V07wFqbeBNTsLHxHa/2lJ/ax2faYP8AVyR2L/x/wPvk/gkj+TY++vpib9pv4JeCfHVn8CF0zTovCMVodLnuWWOTRYLrP/HjJv8A9YeH3ufkDnY5379lXTfjl8HP2528TfC++triJTIbjRbq6xbz36Rx/wDH7Yv/AMs5kO/92/z+X99NjyImn/bgvYacsa3vmp4F8E6b8VPhTZ+LfC+tw61LdHz44VOxMf8APBwPuTofyf5D/fruvhz8RoPENrJ4Q8Xr50zf6Kkl4n/Hxx/qJs9Jv8/fr4N0/VviX/wT1+MktleRf234b1MlxGmILHXYE4+0R9fIuk+Tf/7Omx69/sPHNv8AFCyj8VWl7/adtqf7+Ofydn+xs2fwbNmz/gFZzjyamtGtbT7Z6Jry3Hwn8XXVto2rCX9x8gT968CPwkc/6un+d/CSSPJJI7u8kjv5jySP5kj1JJI8kkju7yO7+ZJJI/mSPJUdZmVQKKKKA1Nzwf4wv/A+rf2hYvvRztubV5PkuU/+Of7dfUXhPxZp3jDR47/T5t8LfI6Nw8T8ZRxnhhXyDW54O8Y6h4L1b7fY5eN/kmtX+5Mn+f46DooV/ZmP/wAFAv2bfFHxPtNP8b+G72/1mbw/asknhhe6E73urVccz9A8fV0QbPnTZJ8Ox6P4wk+EP/Cw08P3MnhL7b9i/wCEg8v9xv8A+emz7/l+Z+78z7m/5Pv1+zvhXxRY+MtDttVsPM+y3AyBKm11/Cpr7Q9O1HS59MubK2n06aF4JrWSJXhdHBDoyHgr7V6mDzOvgf4JpWwUa8ueEj8U9P1y1vLWSd3S3kT78cn8FezfCHR9OvNGj1jTZ01C+uvkeT7n2X+/B/sf7f8A8RX174W/YB+E3h6PxlDNpFxqlpr37u2hvpC50iD7/l2r/fQ+Zl9+d+PLXPyV8NfFDwj4g/Y/+JDeHYLmHUrG+gS7WaV0jk1O18yRE8xP+WLx/vOe/wD44n2+X8Uc04U60Dy61OrhYc9U9muLhLeP7Nb/AOr/AI5P79VPvVneH/EmneKNDg1ewd/sjp/y8fu5IP8ApnJV62uEvIY5oXSSCZN6SR/6t6/VaM4ThzwMySiiiugQUUUUAFFFFABVqCdJ0+zXH3P4JP7lVaKlrmAkuLd7eTY/36jq1b3CTp9muPufwSf3KhuLd7eTY/36hN7MCOiiitQCiiigAooqeCF53/55on35JP4KTdtwC3t3uH/55on35JP4KkuLhPL8mH93B/6HRcXCeX5MP7uD/wBDqpWSTerGFFFFbCCiiigAooooAKI5Hjk3o/lulFFAF6SNL9HmiTy50++n/tRKo0RyPG+9H8t0q9JGl/HJNEnlzp9+P/2olYfwhlGiiitxBRRRQAUUUUAFW7eBNnnTfu4E/wDH6jt7dPLkurl/LgT/AMfqC41eC8k+e6tY0/gj89Pkrjq1YR3YE1xcSTN/cT+CP+5VG+1C10u38+8nS3g37PMkrzf/AIWZr3/Pja/+Ar//AByn2euXvjy6j0jVYEs7R/3/AJlpA8cnmJ/v18bW4qwcaU/Y/GjjqYqHIdn/AMJpoP8A0F7L/vusbWNQ8L6o8k39qQyXfkeWnlzv/wBs6qf8Kv0v/n9vf++4/wD43Ucnw3063gknS6vZJIU8xPuf/G6+DzXP62ZYSpQxNGM4HoZFnlbA4+nPDT5Jf3TGr1D9nv8A5GnV/wDrxT/0cleXx7/40eOT/ponl16h+z6ceKdY/wCvFP8A0clfyDXVk0f6H5xVVbKqkoO97fofT3wx1az0XxtDdXtzFbWsdtMnnTOET+D1qH4ialbat421K6tJUubV9myaF96N+7SuePy5NAw2GrH67P6l9S/vcx+HrAx+t/Wr/Z5TwT43XRh8fBGTfbPpsO9P+Bz157dWnkeW6P5lq/3HruPjz/yPg/7B0P8A6HNXCWeoR2e9Jv3kD/fjrtw0fauNK3Y/b8ufscrpVb9GcXJeT3kf76TzPn/uUyrd5p/2Py3hf7RaP9ySqkcbyfcR5H/6Zp5lf6C4LC4bC4aNHDQ5If3T+BcdjMTjsTKtiZ88/wC8eo/D/wAJ6fcaVpOtuj/b0eSTzPPfy/kkkj/1dd1XPfDuN4/Bemo6PG/z/u5E8v8A5aSV0ANfR0Ye4eQLRRRXSIKKKtQQpGn2m4/1H8Cf36luwC28CRp9puP9R/BH/fqC4uHuJN7/AP7FFxcPcSb3/wD2KjrNJ3uwCiiitgCiiigAooooAKtRzpJD5Mv8H3H/ALlVaKlq4BRRRVAFFFFABRRRQAUUVJb27zybE+/S2ALe3eeTYn36nuLhII/stt/q/wCOT+/RcTpAn2W2/wBX/HJ/fqpWX8TVgFFFFbAeq/DL/kTY/wDr6n/9GVzvxi/5CWm/9eM//oyui+GX/Imx/wDX1P8A+jK534xf8hLTf+vGf/0ZX8O8K/8AJ0K3/X2v/wC3HqT/AIB8e2//AB6p/uV9CaP/AMgXTf8Ar1g/9F1892//AB6p/uV9CaP/AMgXTP8Ar1g/9F1/Z2H+M8+Zbooor1DMt29wk8f2W5/1f8En9yoLi3eCTY/36jq3bzpOn2W5/wBX/BJ/crH+HqgKlFSTwPBJsf79R1qtdQCiiimAUUUUAFT29u9w/wDzzRPvySfwUW9u9w//ADzRPvySfwVJcXCeX5MP7uD/ANDrFt7IBJ508vyYvkg/9DqrRRWqXKAUUUUwCiiigAooooAI5Hjk3o/lulXpI0vkeaJPLnT76f8As6VRojkeOTej+W6Vi4t6gFFXZI0v0eaJPLnT76f+zpVKtIy5gCiiiqAKKKKACiirUECbPOl/dwJ/4/Ut2AW3gTZ50v7uBP8Ax+o7i4eeT/nmn8Ef9yi4uHnk/wCeafwR/wByoKzSe7AKKKK2A+cf2jte+Kek+OtPh8DWF7d6W+lpJcSQ6Ql3H53nz/xvG/8ABs715JN4y/aH+zzCfRdVMe1/M/4pqDH/AKIr658Wf8hX/tglYVxbx3FvJA/3HTy6/iPi/NaWGzvE03hYz97+U/esmyudXL6M1iZQ/wC3j6k/bG8HeIfGXw78GWug6FqOu3cOpJcTwaba+ZIkf2HZvk/4HXyp/wAKP+Jn/ROfE/8A4Lnr3df2uPGCoif2X4f+RNn+ouf/AI9Tv+Gu/GP/AECvD3/fi5/+PV7OU+ISyrDfVacP/Sj53E8KV68+dv8AI534ffCjxxaeEbCG48G61BMjz70ntdkn+skrqLP4c+MrdpFfwhqckD/fTYleN6x/wUd+I+m/Hmx8Dp4c8HnTbi8s7UzSW1353lzpG7/N9p25+c/wV66P2uvGgbP9l+Hz7eRc/wDx6vrMR4nYnCQp+1ow9+N/tf5nmYXhWWI5/Zy+A9p/Z78Jat4UvtcGqafdafFdTWvk/aE8vf8A6yvyc8SeMPj8niLVBaaNqz2v22fb/wAU1C//AC2f/phX6AQftfeL1kUy6ToXlK+/91BN/wDHq8X1S38ueS6T95BdPJOkkf8AtyeZX5Xm/FFPMsTPGzoQnOf/AMjyn2eV5BXwsXSdWUP8J8l618T/AI8+GtOkv9YtrzS7BGVHuLvw9BHGm/6wUuj/ABK+PHiDTYdT0qzvNRsJt2y6tPD0Ekb7Bs6iCvVv2mf+SK6r/wBftqP/AB+SrH7On/JEfDn/AG9f+j5K5fr+G/s/659Vh8fJblOj6jX+v/VPrM/h5vi/vH15/wAE/bzxlr37NfjBvGFncw6/JrWpQQ209klo7J/Z9r5fyIifx76860/4IfEOOxgj/wCEG1vMaJ/yxj/+OVsfDv46a78L9Dn0rSbHSrmGe6e6kkvkm8ze6In8Dp/crqR+134xH/MK8Pf9+Ln/AOPV4mJxeCxnI6nuf4TqwuFx2Aq1PZrnv/McF/wpP4h/9CLrn/fqP/45R/wpX4hnp4G1s/8AbKP/AOOVX+OX/BQb4hfC3wtp2p6d4e8IXD3V4bR0u7W5f5PL3/wTpXWeC/21PHPiLwhourz6P4Zjm1CzS6aOG3uQm/8A7/0VMBhqdCGL5/cmawx2PqV54b2cOeH9dy18I/hN420f4qeEr/UfCWrWFjb6grzXU0PyInlyfO9cF/wUK1L4kaL8avDkvhGxvbzRG8MwGRbTS47sS3X2q6/d73jk2HZsr1q3/av8W/ZzdXOk6DHB/AkcE3mP/wCRq8r+I3xG1T4na9a6tqtvaW15Dax2qLZI+zZvd/43f+/V0MdhcHR5KcOf/EZTwOOx+IVSpLk/wnxT4x+OPxq8NzWz6/v0Q3Ib7Ml1oNtD8iH+APFV+Pxt+0RKkUiaRqbo670ZPDUH/wAYq5+2QwWHwZk4/wCPz/2jX0Zp/wDyCdM/68oP/REdezXx+Go4KhivqsPf5vs9jko4KvUxVah9Zn7nL9opfsF+Ivi3q37SujQ+NNLv7XRv7N1DdJPoqWkf+p4+dI074r6H+P8A8MPGHiL4s6tqOjeF9S1TTp7e2WO7tYvkfbCiPXlXgXxnefD3xPBr+nw21xdQQzQxx3SPs2Omx/ue4r0//hrrxj/0C/D/AP34uf8A49Xj1cdhMbR5KkPZ/wCA644HF4PEe1py5/8AEcGfgr8Qx18Da4P+2Uf/AMcpf+FJ/ET/AKEXXP8AvzH/APHK7TUf2wvGlppd9dw6P4bke1tZ50EkFz/yzjkk/wCe3+xXmnwH/wCCinxF+K0PiF9S8PeErYaebYJ9ltrxd+8yffL3L/3Kzp4DDVaE8RTn7kDWpjsfTrwoezjzz/ruax+CvxDXr4G1wf8AbKP/AOOV6r+0tD4n8L/sI28Wh6fdR+LLO00SD7ALPz50k8+Penl/PWMf2uvGDf8AMK8Pf9+Ln/49XPePf2gfEPxF8MXWganp2jw2ty8LvJapNv8AkkEiffd/SjB4rBYPmqQ98yxeFx+McIVFyf4T4guPif8AHXS9BvJ9a03UrWytovON6/h6AeSP9smP7lYeg/FX45eK7GS90WG51e1STyWmtNBgkRW/ucQ8H2r374wY/wCFR+MTn/mHj/0ZHXCfslNu+Fd/j/oMyf8AomGvcp47D1MDUxf1WF4S5fhOOeCr08bDC/WZ+/H+Y4ubxl+0N9nm8/RdV8ra/mZ8NQ//ABiv1g/aW8J654u+HHh+00TSbzV71L62mngtF/eIn2WT5/8Avt6+PLi3S4gkgf8AjTy69wH7XXjBY0jGk+HvkTZ/qLn/AOPV5jzXC4inOnOnCH+E7a2V4qhUp1Kc5T/xHB/8KV+If/Qja3/36j/+OUv/AApP4h/9CLrn/fmP/wCOV3bftdeMT/zCvD4/7YXP/wAeryPWv+CkHxJ0349WXgNPDng57Ca6soHuXtb3z/Lnjjkf5/tP/TQ/wVzYPAYbGuapT+D3jXFY7H4TknUpx9/3f61Or0j4M+P7bWdMmm8FaxGi31q7vJEPkTz0/wCmlXP+CnWsePNHl+GTeALO5vZppNXF39l0xLvCf6Js++j7P+WldMf2vPGh/wCYT4e/78XP/wAerhfiZ8XNX+LDaWdYtLC1Om+f5P2FH+ffs3797v8A3K0w+OwuDhNw9/8AxGNbC47GVabqLkt/KfF3/CZftF/9AbVP/Cah/wDjFZOj/GT41654nk8OWDvPrkbvHJYx6LamRSmd+/8Ad/wY719b29m95PshT95Xzf4J1CG0/a+8QWdif3b6jrHnTf8APb93P8n+5XtYDGYXGU605YWC5I83wnJjMLXw86MIYmfvy5fiFvvHP7QNrbmzsdG1V0/5bXR8NQfv/wDyB9yvvz9gW68Z61+zF43PjGzuI9ek1jU4IbeaxS0kZP7PttmxERP499eFV6P8Nvjpr/wt0CfSNJstKuYJ7p7uSS+jm8ze6In8Dp/crz6Ge4dS/gQg/wC7E2xmSV6kP3dac/8AFI8m0/4F/EyPTbRG+HPiffHAn/MOf+5XbfDH4N/ELT/EF0974D8QWcZtfLElxZeXH99K9O/4a78Y/wDQK8Pf9+Ln/wCPV578bf8AgoB8Qvhj4TsdW07w94SuJ5737I8d1a3MibNjv/BMlfseD8T62IrU6NOjDn/7ePgK3CNWhSnXnL3D0H/hVvjLj/ik9U/74T/4uvQPFum654d/Yz8awJp91b69b6HqMkVm8O+Tf58jp8n8deU+Df21fHPiLwfouq3GjeG45r+zS6dILe5Cb/8Av/Unif8AaZ8UeL/DeqaHe6ZoqWmpWslrO8CTb0SQY+T56+bz/wAQZ5vReDrwh7k/73949XL+F69GarU5HwNJ4x/aL8z/AJA2q/8AhNQf/GKx/E3xl+NPg/7Mdd8zSFuN/lG90K1j37eHwDBzX1vXzV+2V/x6+Dx3zeY/8g18dlePw+YYqOHnhYL/ALdPpMywNfAYaVeGJn/4EfbtFFFf3ofgBaggTZ50v7uBP/H6ZPO87/8APNP4I/7lQUVHLrdgFFFFWAUUUUAFFFFABRRRQAUUUUAIvSrFneeT5iOnmQP9+Oq7dK5f4j6pr2j+Ebu68PQeZdp/rJI/9ZBB/wAtJI4/43rjxVaNCjKrIfmZPxU+LFl8O7qCC2gTVNj+ZdSeZsjgT/nn/wBd/wDpnXNfFjXvE3ijxdB8NdE8P6tb6peCBZNKu7V7S+vXnj8yNNj/AOrj2ff/AOucm/8AdpXe/wDBPHwz8OvE3xaluvE94174ysz9r8PaTeoPss3/AC0ku0kP+uuo/wC5/B9/5z88P6Xt4X0FvFVtrsml2cniGG1ktINReJDdJbu6O8Yf72zcE46Zr8WxnEeMrOdNfDI6MNhZYqHO5nwT4r/Yh+NlrD8NJtC8dw3esWTxpqF3A/2T+yZvn/0tH/1l0iJ+7w/7x/8AckfZ9k+NfiI/w88O2mny3ceseJWgRQ8kXlo7YAeeRE+4medv4e9em5NeB/GT4U3FtdXfibSTNcxzHff2v35F/wBtP9j/AGK+TrV513zVD2PYewh+5PJ7q6udSvJri6mae6mffNM/33elt7dI4/Pm/wBX/BH/AH6LeNJI/Pd/3H/odR3Fw9xJ/wCgR/3KwPL1C4uHuJN71HRRQAUUUUAeb/tF61Dovwh1r7RoX9twXWy38x/9Rpz/AMF3JJ99Nj/c8v8Ajr3/AOFM0nxQ/Ythi0a0uPgZazaNJb297bGOOGyhTn7XC0nz+RIAz+Y+yTZI7hw2ySvDvjZZ+LtU+HOp2Xg2GOe+uo/IuoIE33c1q/ySR2ifxyf+P7PM2fPW5+zj+0d4f/ao8A3PwZ+LUgu9R1KEw219v8r+10Q+YnKn93cp5YfH8Yj3j+NBvH4B0ZwhVcC5/wAE7f2evB8PhlPiS2o6Z4o1+O4msbJtOd3t9JCAxvsDxo5mkTD73Qfu5E2cSO8nE/tbfsyeDfB/xu8Fah4e8dab8NrjxRqwM0UjyRvps2S/2618vhPnGz59ib5I/nHz1Y+Ev7N/xs/Z1/acXT/BUy3nhHUI/MvtbvUI066sUOPLukTH+moXPllP/RfmJVTwX+x78R/j38fvEms/HCFrbSrG68u9kgk2R6wMfu7Wx/uWXlyfO/3/AN5s/wBd5kken2ufnL5f3cKXsj2H/goprj+GfgDY6dd+GG8YWl1cxQT69qaJ5GlSLjZdyeTsdJnPyIY9ifvH3kDbHJyfg/WH1zwjot7NoL+F/PtY9miyf8usf/LOOP8A2P7n+xXC/tZftVT+OryD4Q/BxJG0WJk06a80FC8moyJ/y5WQT/ligT53T7+0j/Vo/md14T/t6Twzpv8AwlV1a3niHyP9OuLD/VvJ/wDF/wB/+DfUTj7hnUlCdaXIa1FFFc4ahRRRQGoV3vwu+F8vjmU6hfxvHoKvnJGxrx/7if8ATP8A2+9cZpslnHqFq+p2013pyP8AvoIX2O6V9eaDqGnapotrc6RJC+nOmYWg4TZjt/doOvC0YT+Mg1zXNF8A+GrnVNVurbRtD02DfNcTOI4YI07n0FeOWv7Y/wAONR+D938QtMvbi6tYrj7DHpUq+RetdYzHAY3Pyb0AfP3dnz+tdj+0NL4Zt/g14tm8caLJ4i8Jx2nmX2nJb+c8sYIPA/g2MFfzMp5ezeWTYWH5EWf9g3Gm3el6bA9u7v5j3Ej753j8z93+/wDLj3pH/wBc/n+/5denl+C+vVOTnNMViZ0JckD6h8P/APBRnx1bJ41kuvD9hrsk0fnaJ5Z8i30+fIQwSfxzwog8zP8ArHfzPuJInk/Kztq3j7xFqPinxRqE2r6jqU3n3N1d/wCsun/9kT/lmiJ8mz5ErWt7dLO3jghTy40qSv0nBcOUcPOE5ni1I1KnJ7eY+O4njgngSd44Jkjjnjjf929dp8K9Q1SPUZLK2g+0aS/7yfzP+XX/AKaJ/wDG64yO3nkgnnSB5EtU8yeSNPuf79dn8L9L1STUZL22ne301Pkn/wCed1/0z/8Atn8FfaQ0lobHqlFFFewc4UUUUAFFFFABRRRQAVagnSdPs1x9z+CT+5VWipa5gJJ4HgfY/wB+o6KKYBRRU8EDzv8A880T78kn8FDdtWAW9u9w/wDzzRPvySfwVJcXCeX5MP7uD/0Oi4uE8vyYf3cH/odVKySb1YwooorYQUUUUAFFFFABRRRQAUUUUAFEcjxyb0fy3SiigC7JGl+jzRJ5c6ffT/2dKpUUVMY8oBRRRVAJ0q3aWaSR+dN/qP8A0OqtcZ4k8SJql1f+F4YJ47ufZAl5J/q0/wCWn+/XkZjmEMBR9rMznUhT9+Y/x147ex1T+xHsk8u6gREk8/Zs8zzI/uVz8nwfTy5E/tRP+ef/AB4//bK6jwn4ffQ9NkgvPJvJ3n8xJP8AWfu/+B1t1+H5lmE8wrc1TY+bxWNnUn7g6OSSOONPPf5P9ujzHk++702ivKPM1CiiimGxynjpvOvtM38fuZP/AENKXwB40PgPULy8jsPtxuLbydnneTs+ff8A3Kj8cf8AH9pn/XCT/wBDSuf61+F8S+9mda393/0k/wBEfC+hDF8E4SjV1g+b/wBOSPrTw7qg17w1pWpeV5H2y2SYwb9+zfUXizW/+Ea8L6tqqRfazaQ+cId+zf8AP/fqv8Ofl+Hvhr306L/0CqvxS5+G/iT/AK9P/Zkr4LT2tjyoUYPFqj9nn/8AbjwPxx4sPjXXE1I2n2HZbJDs87f9zf8AP/4/XB+JIvM+yf8AbT/2nW30rF8Qf6y0/wC2n/tOv1zw404lw3L/AHv/AEmR9F4h0YYXhLE0aWkFy/8ApyJU0/UPsfmQTJ9otH+/HXYfDvT/ALH4xgnhfzLR4J9lx/2zrhK9J+Hfg+60+6sNbe6tZLSa1f8Adx7/ADPnjr+1IQ98/iQ9Cooor3DnCiirUMCbPOuP3cCf+P1LdgFt4EjT7Tcf6j+CP+/UFxcPcSb3/wD2KfPcPcP8/wBz+CP+5UFZpPdgFFFFbAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFSW9u88mxPv0tgC3t3nk2J9+p7i4SCP7Lbf6v+OT+/RcTpAn2W2/1f8AHJ/fqpWX8TVgFFFFbAFFFFAHqvwy/wCRNj/6+p//AEZXO/GL/kJab/14z/8Aoyui+GX/ACJsf/X1P/6MrnfjF/yEtN/68Z//AEZX8O8K/wDJ0K3/AF9r/wDtx6k/4B8e2/8Ax6p/uV9CaP8A8gXTP+vWD/0XXz3b/wDHqn+5X0Jo/wDyBdM/69YP/Rdf2dh/jPPmW6KKK9QzCiiigC1b3CTp9muPufwSf3KhuLd7eTY/36jorJKz0AKKKK1AKnt7d7h/+eaJ9+ST+Ci3t3uH/wCeaJ9+ST+CpLi4Ty/Jh/dwf+h1i29kAk86eX5MXyQf+h1VoorVLlAKKKKYBRRRQAUUUUAFFFFABRRRQARyPHJvR/LdKuyRpfo80SeXOn30/wDZ0qlRUON9QCiiirAKKKKALUECbPOl/dwJ/wCP0yed53/55p/BH/cqCio5dbsAoooqwCiiigD55/aE+PUfwp8bWOknw+dX+06fHdect55JT95Imz7j/wDPOvKv+Gyk5/4os/8Agz/+0V9SeNLO1uNYjeaysriTyI4/MuLVJP8Anp/frN0vTNPk1K0/4lemf69P+XGD/wCN1/EXGGKy6nnuJjVw3PPm/mkfvWR4XMKmXUXDE8n/AG7E+bP+GzIv+hMz9NU/+0Uv/DZSf9CU3/gz/wDtFZP7JscF38QvE32m2guIxpkj+XcQI6f8fUH9+vp3+y9P/wCgXpn/AIBQf/G68PMf7Ky6sqP1W/X4pHVgP7Tx1J1vrP8A5LE+LNW+MQ1H4y23j06SIvs91a3I037Qf+WKIoTfs/2PSvTW/bKhVQf+EMbP/YS/+019Df2Xp3/QL0v/AMAoP/jdH9l6d/0C9L/8AoP/AI3XNXzrL8Tye1wt+TRe/sb0crx+H5/Z4n4/7p88j9smNv8AmTMHr82qY/8AaFW7X9tqO2Do3gd5YJPvRnVcZ/8AIFZ37YVtBax+Dfs1tbW6ut4SLeGOMHBj/uV9C6fpen/2VYf8SvT/APjxg/5cYP8AnhH/ANM66K/9k4fC0cX9V+Pm+1Iwo/2nXxNbC/Wfg5fsxPmT4uftAxeOPBk+hw+H1gtryeGeG8jv/MP7sH5Hj2ff+eofhz+08ngHwXp2gf8ACMfbvsXnD7V9vMe/e7v9zYf7/wClfVH9l6d/0C9L/wDAKD/43R/Zenf9AvS//AKD/wCN1y/2xl/sPqv1X3Pi+M3/ALLx/t/b/Wff/wAJ88f8NlR8/wDFGY/3tUx/7Qo/4bKjxn/hCmx/2E//ALRVH9o6GG2+M3hJIba1t4/sdr+7t4Fjj/4+ZP4K+ndQ0vT4767/AOJXpf8Ar3/5coP+en/XOurFrKcLQoVHhb+0/vSMML/aeIrVqf1n4P7sT44+MHx+X4q6DZaYnh/+yhbXX2nz3u/OL/u9mzHlrXb+Cf2orfwt4L0e2vPCyXEOnW6Wq/8AE02PeOh/ubPkSvpez8P6X5f2q50vT47RP+nGD5//ACHVe8t7K8k+fS9L8tP3aR/YYPLSP/v3WUs2wDoQw88L+7/xG0Mrx/tp14Yn3/8ACfPV5+2p9skMsngsjnAX+0+F/HyKrf8ADZUYz/xRhJH93U8j/wBEV618WrGyh+FPjCRNP0+N49PBSRLKGN/9ZH/sVxH7KNra3Pwvv3ms7O4catIA9xbRyY/cw8fPW1NZVUwM8X9V0Uv5pGM3mdPFQwv1n7P8sTxT41fGpPi5/Y6pox0hdNM2N1z5xk37P9hMfcr0aH9sSO1tLeA+DP8AUwRwD/iZn+CPZ/zzr6J/svTv+gXpf/gFB/8AG6P7L0//AKBel/8AgFB/8brlqZzl9WjDDzwvuQ2983hlePp1p14Yn35/3T54/wCGyo+P+KLIB7nU/wD7RR/w2VHz/wAUWSB3Gp//AGiu6/aSsbK1+DOqyQ6dYwSfbbX54bZI3+/J/sVY/ZysbK6+C+hST6dYzyedc/PNbJI/+u/3K6f+En6l9c+q6c/J8UjD/hT+ufVPrP2eb4Ynm97+2BHfabf2v/CH+X9qtprfeNSJKeZHImf9X/t1598FvjUnwjj1uN9GOrrqfk523PkmPy9/+w+fv19nf2Xp/wD0C9L/APAKD/43R/Zenf8AQL0v/wAAoP8A43XNTzjL6VGdCnhfcnv75vPK8fUrQrzxPvw/unzx/wANlR/9CU3/AIM//tFH/DZUfB/4QzOfTVOf/RFfQ/8AZen/APQL0z/wCg/+N18z+A7WKb9r/X4ZLaCWD7brA8iSBdnEc/GyunBf2Ti6daawtuSPN8UjDE/2phZ0YfWfjly/DEh8Y/tVJ4q8H6xog8K/ZP7StvI8/wDtAvs+eM/c8v8A6Z1hfCb9oSP4X+GLnRpvDp1QS3jXXnreeSVyiJs/1b/3P1r67/svTv8AoF6X/wCAUH/xuj+y9O/6Bel/+AUH/wAbrnhnWAhQeHWF/dv+8bf2Vj6lf2/1n3/8J88f8NlR7c/8IU2P+wn/APaKT/hsuL/oTM/TVM/+0K+k9L0vT5NRtP8AiV6X/r4/+XKD/np/1zr5S/ZRigvPiP4lW4t4LmP+yZT5c8KOn/HzB2fiunBxynFUK+IWFt7P+9IxxTzTD1qNB4n4/wC7E2/+GzIcZ/4Qxv8AwZf/AGmvNdW+Mo1P4zWnj46Ls+z3NrP/AGd9pJLeQkaAb9nfy/TvX2j/AGXp3/QL0v8A8AoP/jdH9l6f/wBAvS//AACg/wDjdcuHzrL8Hz/V8La+j943r5XmGK5Pb4n/AMlPnj/hsqPGf+ELb/wZ/wD2ip7X9r77dcJFD4MjZ3/vauU/9p177qHh+yvNL1aGHS9M8z+zrr/lxgj/AOWEn/TOvDv2RzYnT/GdrBBDdx/6D50txBG+5/3/ANzf/BXRReVVcLWxX1X4OX7Ujnrf2nSxNHC/Wfj/ALsRbr9tCCzhks7XwgGQHEk8epf67/yB9z2rx/w98Xxovxh1Hx3/AGL9pN3cXs408XBTYZ0kGA+z+Df6dhX2n/Zenf8AQL0v/wAAoP8A43R/Zenf9AvS/wDwCg/+N1z0M6y/DKdOlhbc+/v7nRWyjG1+Sc8T8H90+eP+Gyk/6Epv/Bn/APaKT/hsyL/oTMfXVP8A7RX0R/Zen/8AQL0z/wAAoP8A43XzJ+09DDb/ABX8LRw29vbxvY2/7qCFI05upuycV05d/ZWYVnRWFtZX+KRhj/7TwNJVnif/ACWJp/8ADZcf/QlN/wCDP/7RXHfFz9oIfFLw7baQPDx0lYrz7X9oe784t8jrs/1af36+wNQ0vT49Su/+JXpn+vk/5cYP+en/AFzqD+zNP/6Bmmf+AUH/AMbrjo5rluFqRr0sLr/ikddTLMwxFOVGpif/ACWJ8z+F/wBq5PDnhXSdH/4RH7X9gtUtvOGpFN5To+zy60f+Gyo8Z/4Qz8P7U5/9EV9D/wBl6d/0C9L/APAKD/43XzX4otof+GzNLgW2gSD+0NLH2dIE2f6iDjZXoYGeVZjUqN4b7Mp/FI8/FwzLAU4f7T9rk+GJd/4bMh7+DGH/AHEv/tNebfGr41L8X/7HVdGbSF00zdbnzy+/Z6ImPuV9oR6Xp3l/8gvS/wDwBg/+N0f2Xp3/AEC9L/8AAKD/AON1xYbNsuwdb22Hwtp/4zsxGV5hi4exr4n3P8J6pRRRX+gZ/OgUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUf6uiilvuB5H48+C9ze69Ya14Quv7HvvtSTyBH8j7LP5m/wC1wOn3HT/nmn/AK0vij49+KPgf4p6b8U4fFtzquoWsEdml9NAkccMYGHtJ4I/k8id/n/3/AO46JXpJ54plxbQ3dvJBcwpcQTJ5bxyJvjdK+NxnDWDxHPKHuykSotfwzrfGn/BSjVtL8O+A9Q0j4fywT6iPterrrDeXDJCh2PBYyDG93/1m9/uIU+R9/wAn2R8Kvip4e+M3gyw8T+F7z7Xpd0MfP8kkMg+/DIn8Dp0I/wD11+fHi6z8PSeEb+fxj+78PJ/zz/17yf8ALPyP9uvoP/gnnH4Db4daxceG7q+n8STTwHXEvsI8J+f7OiRp8iIE3/7f399fmOd5VDKpckJ8x62FxNWpW9nOR6J8XPhO9nv13Qos2yfvLqwjT7v/AE0j/wDiK8bjk8yPen3K+i/il8WF8I276bpjJPrcqZGeY7ZP+ej8fe9E7186eY8kkju/mSP9+SvmAxXJz+4FFaXhnw7eeLdag0yxRPPm/wBZJJ/q0T+N6teMfB2oeB9b+x3yeZA532t0kfyTJ/8AHP8AYoOf2f2yrYeHdT1LRdR1a3s5p9OsP9dOn8H/AMc8v+Ouk+F3gbS/HN3qCahqYjaKHENrCcTtngT/AO4hIH1Pz/7fo3wBj1o+Hbl7p1OitL/oCf8ALQfMfM/4Bu/9mrnviR8NLrwdfHxT4VSS1hgk86e3g+/bf9NI/wDpn/fT/wBk4pnbTo+5znN6fDcfB3x5ANc09L1F3+XOkY/1f/PaH/b7GP8Ay/jH7a37Leja4q/FD4cTQx6vqMn2q+0WzbH9pv8Af+12mz7lyn+sfH3/AL/+sH7z6v8AD/iDSPjj4ak0rVUjg1eKPeywn7h7TwH+5n/4h8/xeK+JvDN74N1uTTL91knh+dHgf5HR/wDlp/sf7lXCfszOtGHJp8J494f/AG4PilY/su31+ugvfa3ZXMemw+Np0V4I4MYe7mgzveaN/wB3jHl+Y439NklL9ov9qj4nfEL4U+DPD8HhC+8IyeKbbZqc9kCz6hJkp9ltRnfCkiHzCkn7wo+z7m939Z/5Z7P+Wf8Aq/Lo8x/7/wD00q+ZHJetycnOT/shfAbwp+z/AOB2+IXiG8tdS8X3Mf2Upb/vE0v/AKcYE/57dnfv7IK7Xwz4VvvjB4u1LVPsyaTo8k++5ktUxk/880/vyf35Kyfh/wCB38ea19jM6WdhD++upN/7zZ/sf7f+3/BXo/xB+IVp4A0xfC3hULDdwr5LyQp5n2bphAP45n9PxrCU+c64Qhyf3Tzf4jaBpvhnxZLp+lXzXsWP+PVfne2k/wCeG/8Aj+n+sqn4j8G614V+yvqlk9ulym9JP9ZH/wBc5P8Abr2L4R/CP+wtmua5Dv1V/nhhk+f7P/tt/wBNKxf2hbfUo9Q0+aXUYZNHk+SGwQ7JEm/jk/2+P++Mf7dM0qUfc5zx6jzPM+5XdfDH4ZTeOLsXd6Hh0GJ/nf8AjuX/ALif+zv3/wDRc3xkvPDlxrUNnotqsN3Yp9muLmA7LcKn3INn8ez2+59z12I5vZ+5znn9df8ADXx/qHgjWP3KSXmm3Mn+lWn/ALUT/b/9D/8AQMHQtB1DxRq0Om6XD5l1N/f+5Cn99/8AYr0y5+A9/FrNpZW1250WdPMvb44WeN8gFETp82fk/uYO7PRwdOnU+OB7bpGsWHijS472wnju7KZeJE71+bX7ZP7F8nwtmu/iB8OrRx4VXM+p6Nbg79Jx/wAt4P8Ap1/vx/8ALEcj93/q/wBKND0Ox8OabDp+n2yWlnCMJHGOBV05+orSnUnRnzwPTr4aGKhyTPxC8L6g/iiSO1toPM1J/wDl3j/j/wCmkdesf8Krg/4R/wAj7V5erb/M+1/8s/8Arns/uV3H7YX7HF18Jb25+JXwxgmg8ORSfar/AEuw4k0d/wCOeD/p1x/rI/4MZ/1f+r4v4X/FCDx5Y/ZbnZb67AnmPHH/AKu6T/npH/8AG6/XuH86hi17HE/GfN+/Qn7GsdXo+h2vh/TY7KzT9x/HJJ/y3k/56SVct7dLOCOCFEt4IU8tI4/9WlO3Ubq/ROQYtFFFUIKKKKACiiigAooooAKKKKACp4LdLiTY7+W/8FQVPBA87/8APNE+/JJ/BUy2AI7N5JpEb93s+/5n8FSXFwnl+RF+7g/9Dq1JeJfwfZkfy9n3JJP4/wDfrJkjeOR0dPLdKxh+8+MYtFFFdAgooooAKKKKACiiigAooooAKKKKACiiigAooooAKTmrdnZps86X/Uf+h1wXiTxRdahBAnhW6/tC7R/9KjtI/M2R/wDA/wDbryMwzOhl0OeqzOpUhThzzDX/ABH/AG35+j6DdTR6zHP/ANcP3af6z95U/h/w+kdraXWq2qSa6knmPdyfvJP+mf7z/cpfD/huC38jVJoJo9amTzLqSR/+Wj/6z93W9X4fmGYVsxre1q7Hy+KxU64UUUV5ZwahRRRRqGoVkap4s07R76OyuXmjndN/7uB5K168w+IH/I3Qf9cIP/RklY16ns6ftD0ctw9PGYunh6nU1vEmqWusXFo9nv8ALhSRH8xPLpfC/hTVPF11cWukwLcT2yee8bzInyb9n8dY/meWhdzXsvwV8J63oOvatPqGk3llBNYxpFJMv33376/njMMdPF154mfxn+kuDwWG4RyOngcFP3YfDzf4uZ/y/wAx6V4L0+XRfB+hWF1FsurazhhmTfv+dEqDx7ptzrngnW9OsYke+uLbZCsj7N770r6N8I/C/wAL6t4V0i8uNPdpri0hmdxczJuZkGf46h8cfDLw1ofg/Vb+10947mGB3RzczPtP0311/wCq+NhR+tc8f5v690/E4cUYb62rQlz83yvzf4j85vEfhnUfCepf2dqkCW9z5KTlFmRzsf8A3P8AcrmNXs5rjyPITzNnmV7R8Z/Cut6t4ul1C00m4utPhsU866jX5ECb3evKPuxgZ71eR5xWyfH08fh/jh/8jyn7pisNQ4qyZ4TEz+Ll5uX/AMC/vHI17j4L/wCRR0b/AK9Y68Ojr3HwX/yKOjf9esdf6B4CpCtFVI9T+BKkeWTS6G1RRVq0gTZ51x/qP/Q69Vy5Uc4tvbps865/1H/odR3M73D/AD/c/gj/ALlT6pvkkSb/AFkD/wCo8v8Ag/2Ko1lD3vfYwooorcQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVJb27zybE+/S2ALe3eeTYn36nuLhII/stt/q/45P79FxOkCfZbb/V/xyf36qVl/E1YBRRRWwBRRRQAUUUUAeq/DL/kTY/8Ar6n/APRlc78Yv+Qlpv8A14z/APoyui+GX/Imx/8AX1P/AOjK534xf8hLTf8Arxn/APRlfw7wr/ydCt/19r/+3HqT/gHx7b/8eqf7lfQmj/8AIF0z/r1g/wDRdfPdv/x6p/uV9CaP/wAgXTP+vWD/ANF1/Z2H+M8+Zbooor1DMKKKKACiiigCa3t1nk2O/lyfwUsdo8kzo/7vZ9/zP4KLe3e4f/nmiffkk/gq9Jdpfw/Zkfy9n3JJP4/9+uec2noMqTzp5fkxfJB/6HVWiSN45HR08t0oraKSWggoooqgCiiigAooooAKKKKACiiigAooooAKKKKACiirUECeX51x8kH/AKHSbsBVooopgFFFFABRRRQAUUUUAePfF74ueFfh/wCJrfT9f1C6tLqeyjuo0jtXcbPMkT/2nXG6f+018OLe+gd9avNkbpJ/yD37V23xZ+FfhLx74mg1DX9Jkv7qGyjgR/tTp8nmSP8Awf8AXSuMX9nP4aKcjw23/gwm/wDi6/iLi15Qs9xLxHPz8390/eskWaPL6KocnJ/28fPv7O/xA8P/AA/8Ya9f69ezQW11p8kELxweYXczRyD/ANAr3g/tL/Dcf8xe8/8AAB6tf8M5/DT/AKFpv/BhN/8AF0z/AIZ0+Gh/5ltv/BhN/wDF189jcdk2Pq+2q891/hO7B4PNsDS9jT5P/JiD/hpf4cf9Be8/8F70f8NL/DfOP7XvM/8AXg9eIa34B8P2n7TVh4Sh094/Dk2o2UElp50n3JEjL/P9/wDjNe7/APDOPw2LZ/4Rts/9hCb/AOLp4rB5NhIwnU5/fjzfZFhsVm2K5+Tk9yXL9o8S/aR+KHhn4hf8I4NCvprz7D9pM/nWrRbN5j2fXhDXsVp+0h8PIdOtEk1e88xLaCJ/9Af+BEH9Kun9nP4aH/mW2/8ABhN/8XTf+Gcvhr/0Lbf+DCb/AOLoqY/J61Cnh3z2p/4S4YXNqdadf3Lz/wARW/4aY+G//QXvP/AB6k/4aY+G4/5i95/4L3rxHwl8P/DmqftKar4TutO36DBeajAtr58nypCkhT5/v/wCvdm/Zz+GuefDbf8Agwm/+LoxuDybBzjTqc95R5vskYXFZti4TnDk9z3ftHgfxs+JHh7xh8TPDmsaReTz6fY2tsk0klvsdXWZ3f5P+B19DL+0Z8OJpZ7+41fUI7F5n2f8S9/33+wlFp+zT8MWT7Ve+G5o7T/sITb3/wDH6sa1+zp8O5ZI5x4fd7Tbtg/4mE+xE/uffoxGNyjE0adOcZ2p/wCH/MWHwmbYetOcOT3/APEZN5+1F8PLyXc+r3kaINsccenv8lQf8NL/AA3P/MXvP/AB68z/AGkPhT4T8BeD9HvtA0p9Pu7jUJIJH+1SSB4wmf467nwD8A/AWteA/Dep33h9ri9vLBZ5n+2TIHfP+/TqYPJ6eFhi3z2n/hCGLzapiZ4Vcl4f4in8RPj94F8RfDzxJpdhqdzLe3ll5EKPZum996H+hrk/2e/jB4S8A+Cb3S9d1K5tLqTUmuI1htWl+Qxxr/7JXqo/Zz+Gi8Dw23/gfN/8XSf8M5fDT/oWm/8ABhN/8XSp4/J6eEnhFz8n/boTwObVMVDFe5z/APbxW/4aY+G4/wCYvef+AD0f8NL/AA36/wBr3n/gA9ePftK/DPwz4A/4RtvD+mtYfbTcicfanmD7DHsPz9OHr2a1/Z3+HM2nWLyeHG8yS2glk/0+b+NE/wBv3NFbB5PSoU8Q+e1T/CaU8Vm9SvOguS8P8Rwfxw+NXgzxp8NNR0fStSuZ7+e5gljilsmT7jOXO8/UVJ8Ffjd4L8G/DDS9F1XU7qDUYJ55Hjjsnf775T567HV/gb8JvDumvf6ppCWFkkkaPcTX91sTf71wfjrwz8E7PwLrlzoV3YPrKW26zWG+umfzN6cbHHpmujDyy/E4b6jTp1eTn393+rHJXjj8PX+t1J0ufl/vHe/8NMfDf/oL3n/gA9H/AA0x8N/+gvef+AD18RfL6/oaOP736V7f+qeA/mn9/wDwDxf9acb/ACQ/r5n2/wD8NK/DjOP7XvM/9eDf414d4R+I2gaT+0drHiy5u5otDurvUZkuEgZ5NkySbPk/4GK5/wCA+meC9T8VX8XjeeGLS1sXeDzrh4QZ/Mj7pz9zzOK9/wBF+G/wP1/Uo7LTFs9SvpA2yGHUbre4Qb37elcksPgMn9rR5KklOOvXT8DshXxubclbnhDkkag/aY+G7dNYu/8AwAeo/wDhpj4b/wDQXvP/AAAesr4jfAvwF4e+HfiXVdO0Fre9srMTwv8AbJpAj+Yg/v47muQ/Z5+EvhDx14FvdS17R3vr1dSaBG+1SR/II43/AIP9+vGp4PJ6mEni1z8i/wAJ7M8Vm1PFQwvuc/8A28emaf8AtN/Di2vYJH1q88tJEf8A5B79q+f/ANnvx9oHgDxtrF/rt1NBZ3WntAjpDvLP58L9PojV9Bf8M5/DT/oWm/8ABhN/8XS/8M5/DQf8y23/AIMJv/i6eHzDJ8LQqUKfPap/hM6+CzbEVoVqnJeH+Iq/8NLfDj/oL3n/AIAPU1r+0Z8Pr6aOG31S/kf2sHqxb/s0/Du8m8uHww+//sITf/F1414k8I+HND/aOsfAWl2BTw1cXunw3kXnvvnSRI3kTf8Af2fP+lLBYHJsY5qnz+5Hm+yPFY3NsJye05Pf937R7Brn7THw5/4R7VrCx1m8d57K4t/MOnuPOkdHCf8AAORXif7NvxP8M/Du38TJr97NZm+a2MBhtWl3hBNv+n30r2pf2dfhqyc+G2z/ANhCb/4un/8ADOfw0/6Ft/8AwYTf/F0U8fk9HD1MOufkn/hFPBZtVrwr+5eH+Iq/8NL/AA36/wBr3n/gA9H/AA0v8N/+gvef+AD1Y/4Zx+G3/Qtt/wCDCb/4uvCvCfgDw7qn7Sur+E7jTvM0G3vdRgS18+T5UhSQp8/3/wCAU8Lg8mxcJzp8/uR5vshicVm+FnCFTk9/3ftHuA/aY+G5/wCYvef+C968F+O3xG0Lxx8QtD1PRbuaeys7SGKV5Lfy8uk0jn5P+BCvoH/hnP4af9C03/gwm/8Ai6T/AIZz+Gn/AELTf+DCb/4ulgsdk2AqutS59f8ACPGYLNsdSVGpyf8Akwy8/ac+HFxfTumt3mySR3/5B796hb9pb4cL11e8/wDAB6tf8M5/DTr/AMI03/gfN/8AF15b+0b8KfCXgXwPp2o6BpL2F1NqHkO32qSQFNjnHz/SowtDJsdXhQp893/hNsRWzfB0ZVp8n/kx6QP2lvhw3TV7z/wAevEdb+Jfh28/aWsfGEV9NJoMN5ZSvM8Db/LjhjR/k/4Aa9b+H/wF8Ba14B8NanfeH2uL29sFnmf7ZMgdz/wOt3/hnP4af9C03/gfN/8AF10UcVlGXTqKHP8AyfZOethc0x8Kbnyfz/aKv/DTHw3zj+17z/wAeg/tLfDhf+Yvef8AgA9Wv+Gc/hp/0LTf+B83/wAXXiP7TXw38NfD1PDb+HtMbTzefaRL/pLzB9hj2ff9nrLBYTJswrxoU+e//bprjMVm2AozrVOS3/bx9/UUUV/fx/OoUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAITimahqGneG9HuNa1uf7Hptr/y0/wCWk8n/ADzj/wBupLy8sdA0e71vWJ/s+mWqeY7/APLR/wDcrxnwz4d8XfthfFiz8PaW0WkabErzB5mzBpdij7HnKf8ALaf50j2f9NMfIm96+UzrOIZdT5YfGRKX/LtfGTeEfCXjX9sz4qRaLokf9maPp/zzXUib7TRrV/43/vzyfwJ/H/uI71+pHwl+Efhz4K+CbLwt4Wsja2MJ3yTON89zMf8AWTzSfxu/GT/7LwKHwP8AAPgb4S+E28FeDTbKNJmRdSjWZJLtrt40fzLoj/ls6bH5x8hTYAmwV00fj/wzJ4Zk8TR+ItJfw5GrvJq/2yP7IiI5RyZt2zCuCMn0xX4hisVUxdTnme/g8NCguefxnzv42+H+s+FfEn2eY3WrrqE3+i3n33un/uSf7f8An/cqR+A9fk8USeH/ALA39oonnZz+72Zxv8z+5mvqTUtc07R5rCG91C2tLjULj7HZpNMiPPPskk8tP777I5H2AfwOegrSnV2gcQuqPt+SRl3Vwmn1Omz5F8H6R9t8YabZX19N4fnW6A858JOjp/An9x3+5X1J4o8L6d4s0abTdRh82CTkEYDo2Dh0OPvV8seMLDWbTxJe2viQPPqk7+Y74xHc/wDLOPYP7lenfC/4q3FjqC+G/E0kkdxHJ5EN1dH51fOfIn9/7j9/yLsww0oQ/dzOds59a+BPiqS0u431DRryTf8AJ9y5T++n9yZMDjoR/wB9p9DaHrlj4l0uG/0+4S4tJhlJI+hqp4o8L6d4t0WbTtQh8y3k6HgOjYOHQ44avn25XxJ8Edcu7W3mWXT7xH8maRCIJ/8AbOPuTJ+R/wDQEdH8D/AdB8SPhndeE9RfxR4WSS3jhk86e3tcb7d/+eif9M/78f8A7JkV5NcXD3k891NO9xPM+955P9Y8ldNZ/EzxPZ6Jfad/atxIl0277VJJ+/T+/wCW/wDt/wDjn8FcrQefUnB/AFFFFBnqT6bfXOk6hDeWU00N7CMI0P3zXunwj+EZ0Upreux79Vf54YJPn+z/AO23/TSvBP8AP7uum1L4jeJdSk0ySXVZopNO/wBVJb/J5kn9+T+/8nyfiaDShUhT+M+kfG/jbT/Amjte3zeY7/JBaxn95M/9xPevEvC3hvVvjR4lk13XZHTSo32Yj+5/1wg/9nk/yieF/DerfGrxLNrOtTSR6VC/ls8YKf8AbCD/ANnf/KfQMcNloOmpHFHFa2FumAB8iRIn+FM9D+N78/gIb+wsrfw/PZnZYaelu0WYW8nyY9v98fcx618hXdnbR3s8GkTPeWKP5drPImzeleh+L/GuqfFzW18O+HlkbSt+TnKm5H/PZ/7sH+euxDe1L4MpYeINH0aLxBaxwX6fvvPkSO7fZ9/yI/4//af+3QZVv33wHoPwk0fQdL8Kxz6BdjUBPzc3zr+8eQdn/uFP7nauH8XftjfDfwb8YdP+H2palJHf3H7q5v0wbTT53I8qGeT+B35/3Pk37N4r0bxH4LuB8OdR8N+FNXfwddzWbw2eqWsCTvZv/wA9gj8OfrX4zad4dSw1TUjc6jp/iCe2vZ4Pt9jdfarS6dJP3k6PJ99JPv7/APppvr08twE8xrexgFetPDqEIH1f+0t+3x4gvvH1rovwl1COLS9Hvd8+rmNJ49XmT/l3QdPsvrIPnk/geNPnfCuv29PibcfG6x8QR6baweFoLUwN4Xjn3wTRn/WTvdeXv87eBs+T5EGzy/nd68At9PtbO6nnhTy5Jqnr7zC8M0eT9+eZ7SvP35TPY/Cf7dHxj0u98czX8en6vJqySSafBOmyHSbrISPyD/HBsA+R/vuN+/533/PfhvS7qO+j1eZ3t50n8+CSP9xJ5n/PT5Puf9c66H/yJWn/AMI/qP8Abn9kfYpv7S/5969XC5BhsPU9oc84+0+OZ6f4L8cJ4og8m52W+rInzx/8s5v+mkf/AMRXU14b4T8P3viDWI0s3ez+yv5k93/z6/8A2f8A0zr3KvuaMpSRpqFFFFdRkFFFFABRRRQAUUUUAFFFTwQvO/8AzzRPvySfwUm7bgFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6Li4Ty/Jh/dwf+h1UrJJvVjCrsciX6eTK+ydPuP/AOyPVKitJR5hBJG8cjo6eW6UVejkS+TyZX8udPuSf+03qjJG8cjo6eW6VnGTejAKKKK2AKKKKACiiigAooooAKKKKACiiigBM4qxBbpHA9zcP5don/j9PtbdPL864/1H8Cf364Xx348ey1j+xHtYfLvYI4/M8/Zs8zzE/wBXXjZhmFLA0nVmE5+z9+ZJ478f6hpM1o+lJa3kbpJ+7jR59n/PP7lHhvwna+H55J7ae6kkmg8t47jZ/wBdKk8J+F/+EPju0hvftHneX/B5f3PMrbr8QzDMK2Y1uaWx8nisV7cKKKK8zU4NQoooo1DUKKKKNQ1CvNPiBG8ni6DYj/6iD/0ZJXpdcT4yz/b/AB/z6x/+jJK8HOsZ9SwU6p+meHeQw4iz2nhJz5Pt/wDgJz9581jMP9h/5197fDbwja+Lr+4tbmWaFIraB18l0r4JvDixm/3X/nX6L/AI517UP+wfBX4hllGGIx9GlW+Bn9m+IlSeHy+nKm9ub/209i0XSY9H0m1sY3Z4beFIUZj8x2cc/lTfEGhw+I9GutOnZ0huI/Ldozhq1CvHNAHFfu/sYez9jb3T+VOeXPznyj8bPC9v4UtdasLZ5ZlOkSzbpjl/uTf/ABFfE3/Lp/wCvvb9pg/6XrP/AGAZR/45NXwTb/6lP9yvwfMKUMPjasKfwc5/WHh/UnWy6TqPdR/U5640/wCzx/Jvr2LwjcQHw3pMCTw+elrH+73/ALz/AFdebXlWPDckHhe+/wCEhuUjkgTzI0j3+X58j1/TfB/FFZ0IVanwf/bH8f8AFmUf6tZxUy7n54R5f/Jj2m3gjWMTXP8AqP8A0Oobi4e4k3v/AMAjj/grL0HxBJ4s0a31R4Et/O8zZHG/mRpsk2VfP3a/oDD1YV4RrQ+0fOli0vPI3o6eZA/346Li0+z/ADo/mQP9ySoKntLzyN6OnmQP9+OuhprVCIKKnntfs/zo/mQP9ySoK1TTV0AUUUUwCiiigAooooAKKKKACiiigAooqS3t3nk2J9+lsAW9u88mxPv1PcXCQR/Zbb/V/wAcn9+i4nSBPstt/q/45P79VKy/iasAooorYAooooAKKKKACiiigD1X4Zf8ibH/ANfU/wD6MrnfjF/yEtN/68Z//RldF8Mv+RNj/wCvqf8A9GVzvxi/5CWm/wDXjP8A+jK/h3hX/k6Fb/r7X/8Abj1J/wAA+Pbf/j1T/cr6E0f/AJAumf8AXrB/6Lr57t/+PVP9yvoTR/8AkC6Z/wBesH/ouv7Ow/xnnzLdFFFeoZhRRRQAVPb273D/APPNE+/JJ/BRb273D/8APNE+/JJ/BUlxcJ5fkw/u4P8A0OsW3sgEnnTy/Ji/dwf+h1VoorRRSAvRyJfp5Mr+XOn3JP8A2m9UZI3jkdHTy3Sir0ciX6eTK/lzp9yT/wBpvWX8L0GUaKJI3jkdHTy3SitxBRRRQAUUUUAFFFFABRRRQAUUUUAFFFWoIE2+dcf6j/nn/fqW7ALBAnl+dc/6j/0OoLi4e4k3v/wCP+5RcXD3Em9/+AR/3KjrNJ7sAooorYAooooAKKKKACiiigDhfG+r6fp+uRwXeqafZzvBHJ5d3ewQSeX5kn9+Sue/4SLRv+hg0X/waQf/AByuW/aH+Adv8WPGVjq8mvyaQbbTo7XyY9P8/f8AvJH3796f89K8wt/2LbW4uYYB44kG99gxov8A9vr+IuLsLlVXPMTKvieSfN/LI/d8mxWZRy+jCjhueH+I94/4SLRh/wAzBov/AINIP/jlH/CR6N/0MGi/+DSD/wCOV8Y/B/4Sx/FfxJqmlyas+lrZ2rXXnJbeeWxNHH9zen/PSvUv+GMLT/oeJf8AwU//AG+vmsTk+W4Wfsa+KtL/AAnqYfNMwxUOehhv/JjG8SahaD9sPTL5Lu1ksF1LTpPtUdzH5PEMOT5n3MV9K/8ACRaN/wBDBouf+wpB/wDHK8D/AOGMbXGP+E2l+v8AY/P/AKPoX9jG2Vs/8JzL/wCCn/7fXRjZZPjI04PE/BHk+GRhg/7Uwkp/7N8cub4onvn/AAkWjf8AQwaL/wCDSD/45R/wkejf9DBov/g0g/8AjlfHPxr+CsXwj/sVotZk1camZuZbPyPL2bP9t8/fr0eH9jm1urW3mPjWVPOgjnGdJ7PHv/571hPKctpUIV54r3J7e7ubwzTMKladCGG9+H94y/AupWcf7XWt3z3lnFYyX+rvHdSXUaQfPHP5Z3/c7ivqK01TQvJ+1Xmu6THa/wAH/E0g/f8A/TP/AFleB2P7FNjHGbq78aTi0Q8IukbZJfp+/qLUP2RLfU7gTN40aI/cVE0XYip/3/rqx0soxtSFR4n4Y8vwyOPBf2nhITh7H7XN8UT3q88V6TeSb5Nd0X/nmkceqQfJ/wCRKfZ+LtFt/MR9d0WSCT78f9qQf/HK+Kfid8IIvh7440fQI9XbUotRhimF1LaeSU3zPH9wO/8Acr1O4/YttIJHjPjeYmN9mf7G/wDt9ctTKMuw8ITqYqynt7u50080zDETnCnhvg/vG5+19PYT+AfD/wBg1Ox1SBtUeRXtbmOT/liP7jmu8+GevaTH8M/CkE+taTbyR6bDHJHPqEMcic+715hpv7JFtYlopvGclxZS/wCuhOkZ/wDa/wB+mXf7Edtb7HTxxO9vJ9yQaLj/ANr111JZPUwUMH9Z0h/dkc1P+06eKnivq3x/3onuX/CRaN/0MGi/+DSD/wCOUf8ACR6N/wBDBov/AINIP/jlfNvjD9lS38J+EdW1xPFsl4dPt/PMH9mbA/z4+/51YHwj/Z6h+KXhe41eXxFJpXk3jWogTT/PLYRH37/MT+/09q5YZRl1Sg8QsT+7X906/wC1Mwp1vYfVvf8A8R1P7YGpWeof8Ih9j1Gzv9i3If7JdRzhDmPqUr6B0/xFo39l2H/E90b93ZQR/wDIUg/55x/9NK8L/wCGNbQAf8VtPnvnSP8A7fXIfGn9nUfCbwxZayut3GopcXgtBHNpv2Yj5HfeH8x8/c6V3xw+V5jSoYClibyhzfZkcTr5lgJ1sbUo/Hy/aJvjh8ctU1y58S+DFi0uTSIdTeOK6td7SOkMjiNw/mbCMH0714GcnvTuabX6DhcLRwdH2dNWPz7FYqtip89VjaKKK6DjHLxXX/Dnx9e/DjxPBrlhbW9zdwJIsaXQdk+dChzsdT0f1rkPek3YqZRjVjyyNYTnRlzQPs66+IkXjr9m/wAQajql9pNpq95ZzR/YYLqNJPknj2fI8heqX7Kmrafp3wyvobvU9Ps2OrSSBLu9ht3I8mHp5jivmj4e+FD468caJoH2r7H/AGldR2wmWHzNm84zsHWvfpf2K4of9f4wuI3/ANvRf/t9fC47C5fgaFTB163Iqkuf4fwPucFisfjq0MTRo87hHl+I95/4SLRj08QaKf8AuKQf/HKns9X0u7n8iHXdFkk/7CkH/wAcrwfS/wBiSDU7yG1g8a3HmSPsz/Y3/wBvrzT4X/Dez+I+vat4bt9V/syG1tnuZtQWzM0lyUmjQJs3psT5/wBOf9jxKOT5diKc61LFe7D+6ezWzTMMPOFGeG9+f94+z5/FWhW8L2tn4g0bZ/y3n/tSD5//ACJ9yvlrxVqFnJ+2HpN6t9ZvYR6hpbPeJcx+R8kEAc+Z9zHBrV/4Y0ts5HjeUH/sE/8A2+l/4YxtduP+E2l+v9j8/wDo+t8FUyjBTnUWJvzx5fhkYYyGaYuEIfVvglzfFE98j8R6L5f/ACHdF/8ABpB/8co/4SLRv+hg0X/waQf/AByvn+6/Y9tLPTL65PjSWT7LazT7BpON/lxu/wDz39q87+CfwVh+LkWtyTa2+jjTDDnyrPz/ADN+/wD6aJj7lc1PKctq0J14Yr3Ib+7sdc80zCnWhQnhvfn/AHj7E/4SLRv+hg0X/wAGkH/xyvmzwPqNlH+15rt+93ZxWMmoavJHdSXUaQfPHPsO/wC5jkVrf8MY23X/AITmXP8A2CP/ALfS/wDDGFr/ANDvN9f7H5/9H10YJ5Ng4VoLFfHHl+GRz4pZpi5wm8N8Eub4onvn/CR6N/0MGi/+DSD/AOOUf8JFo3/QwaL/AODSD/45Xgf/AAxhaf8AQ8S/+Cn/AO315b8VvhHH8NfF+laLHq76nFe28U4uZrXyCu+R0xs3v/c9a5sLk2W4yXJh8Vf/ALdOjEZpmGFjz18N/wCTH2d/wkWjf9DBov8A4NIP/jleN/tX6tp+ofDfTILTU9Pu2TVvMK2t7BO4HkvyfLc1lXH7Ftrb3U0J8czExvsz/Y3/ANvqMfsY2vfxw5+ukf8A2+tMHTyfB14V1ifg/uyM8XUzbGUZ0Xhvj/vRPXPhjrukw/DLwpFNrekwSR6bDHIk+oQxyJz7vXQ/8JFo3/QwaL/4NIP/AI5Xga/sZ2it/wAjvP8A+Cj/AO315jqnwdj0r42WfgBdYaVJ7m1gGpG2CHEyI4Pl7/8Ab/v1Ucty7MKlSpRxN/t/CEswzDAQh7TDf3PiPsr/AISLRv8AoYNF/wDBpB/8cr54/a/1Kzv4vB7WeoWd/wCWlzv+yXUc4TmPg7Ktf8MaW3/Q7zf+Cn/7fSj9jK2Bz/wm8xP+1o+f/a9Xl/8AZGX4mNdYn/yWRGO/tTHYaVH6tbn/AL0T7Xooor++T+ewooooAKKKKACiiigAooooAKKKKACiiigAq3Bbp5fnXH+o/gj/AL9EFunl+dcf6j+CP+/UFxcPcSb3/wCAR/3Kx1logEnuHvH+fZs2eX5f/LPy/wC5XmWmfCKbwr8UNG8TeGdXm0C3sJvtSC0f9/bP/wA80/5Z+XJ/00/g8xPnr0qivOxmXYbHLkrQA828PfDvWfCuqePb9vHepafb+ILWaC6voJtk91HNJ5jyXT/xv/1z+f53+dN+yvKre11aDwHdeEItZmOgzXv9ovpSfJYvMnyI+z/P/LP93+7r6X1DT7XVLWS1vII7iCT78cleZSfCu9/4SD7LDP8A8Sl/3n2+T+CP/nns/v14s+H8FT+CA7R7nnHidvEGtaP4chuNd1LV08M2v2XTIJ7l/wDQ037/ANx/c/5Z/wC35ccf/POOv0u/Ye+Knjn4rfCU3vjbSZ4zbTeRp+vzYT+2oSMmby+2z7vmD5H6pxX5pfECS68D3UmnXP7u72eYl3/yzeP/AJ6R1+ln7DHgv4j+Bvgzb2/xAumCSFH0TRruP/TtLs9mVhnfPqfkgx+5TCbz9yP834goYbD1uTDHZgP94fKdj8YvFWoeFvFWgXg0Czu7K2Pmx3l1Hvdn/wCWkcb/APLF9nTP38/7FZHxYbwl4w8H23iizvVg1F8wRrs+e5/vwSJ1yn/jle2avo9n4h0yaxv4VntZk2SRuOorjNR+DOg3Hg2PQoEe3aH95DfN88/nf89H/vn618ievOlL3zivhH8XTb+RoWvzfJ9yyvpDjP8AsP8A/F122t+IvCHjS91XwleXMMr20ZebzPkRNn3yj5+/H3x9yvMrH4B65NpeqPeTwW99E5Syt45N6T4/5aO/8G+vM7izeznntbm1e3nhfy3gkT94klBye2nThyTJ7qO2t7ydLS5+32qP5cN1s2ecn9/ZUFFFI4gooooHqFaXh230y81qyg1m6ez0t3/fXCf5+T/rpWbRQB7R8SPilD4ZhXw14T8mCS32xzXMCp5VqP8Anmg/56f59SnH+NPidq3j+xtNPZFsLVI43uoIZM+e/wD8R/sVxdnb/aJ44EeG33v5fmXH7uNP+mkn+xX054O+GmjaH4Xn02SKHUhfoPts8iA/aev/AI5/dpnXT5655d8F7vV7XXb220a2hnszATeTXIKr5wj/AHPzj8tn9wh+9W/Bvw317xt4ol13xck1r9nn+dfuPM6Pwif3IE/8fr2nQPD9h4T0eDTtOgEFrCMKOv8AwJq2AT1NB2Qw9o2mc14+8D6V8SPBur+GNbhe50fVrZ7W7jjmeFnRxggOhBFfkv8AGv8AZ08Tfs4/ERNMZ/7T8P3+99M1U/JHcwoMvHP/AHJ0GP8A0NP7ifsb6V+bf/BQ74M+ONJ8dP8AE+PU73XvCxgSAJzjQMf7Cf8ALGT+/wD3/v8A8Fehl1f2FeEzkzKjz0+c+cK7vwH4D/tDy9U1VP8AQP8Alhb/APPf/po/+xXnOl3j6hYxzvB9nkf/AMfrt/h34kvdL1WPToYJNQtLp/nt4/8Alh/00j/9nr96ozhUUJnlc/PA7vw/4D07w/q0+opvuJP+XWOT/l1rpNn7zf8A8tNnl+Z/sUUm0V68acIbGYvlpH5mxPvv5j/7dFFFagFFFFMAooooAKKKKACiip4IXnf/AJ5on35JP4KTdtwC3t3uH/55on35JP4KkuLhPL8mH93B/wCh0XFwnl+TD+7g/wDQ6qVkk3qxhRRRWwgooooAKuxyJfp5Mr7J0+4//sj1SoqZR5gCSN45HR08t0oq9HIl8nkyv5c6fck/9pvVGSN45HR08t0rOMm9GAUUUVsAUUUUAFFFFABRRRQAVbt7dPL865/1H8Ef9+kgt0jj864/1H8Ef9+uF8UeLH8UST6Ro889vq0M/wDyz/cRpGn+s/eV4mYZjSwNLnmROcKcOeYeL/FUevX1/wCFkgeO7uvLtEuP+WCf8tP9+pPCfh+Tw/pslrcyQ3Ejz+Z5kf8A2z/v/wDXOjw/4f8As9raT6lawya0jvI93J+8k/6Z/vP9yt+vxLMMwnmFb2stj5bFYr28wooorzNTg1CiiijUNQoooo1DUKKKKA1CuK8ZZ/t7j/n1j/8ARkldrWfqmn2VxBd3U1rDJOkEmySRP3n+rrwc6wc8dgp0YH6Z4d59R4dz2ni60Oe/uf8AgR5zdf8AHjP/ALj1+in7P5C65qGT/wAuEH8zX51yI9xYyJ/y0dK+q/h98VLTXbi7ttC1DULS5it0eV/JeH5Puff/AN+vwzC4j6liaWIf2D+1+O8FWx+CUaf9fCfb+7dQzBeDXP8Agu5kuvCeizzyNNM9lC7ySfeZigyT71V+Il7NY+DtZmt5ngnS1d0kjPzqcdRX7rLFQjhXif7vN+B/KEaLlV9j8jxf9pfDXesn/qBS/wDoE1fBcP8AqU/3K+mPil8ULC1bUtK1W8vZtTuNOdIpHheY7HR0T5/9+vnS3s0jsftV3/qET5I0+/M9fheIq/XMVOtD7Z/WPBeFnl2XSVdaLl/VkIs0+zi6vf8Aj13/ACR/89qNL8QfZ9Y+23MHmRokkaW8f8FZtxePqE+9/wDrmkcf+rSoOa/dsnwE8uwvsZn8WcbZ9S4kzipjqMOSEuX/ANJOkk09/wC1f+EyTZ/ZqP5/2T/lv5afu/8Acrt/DfiCDxRYyXUMD28aSeR5clZPhe3gvPCMEFyiXED+Zvjk/j/eSVlaxoeqafqkc+j/APEv0WFI557e3n8vfs/1n7v/AHK++yPOZ5ZP2cvgPicLivZ+5M9AwDR92snw/wCJtO8T/aJNPkn/AHPl7/MTy/v1rZr9rw9aGIh7SjPmie/6Fi0vPI3o6eZA/wB+Oi4tPs/zo/mQP9ySoKntLzyN6OnmQP8AfjrZprVAQUVPPa/Z/nR/Mgf7klQVqmmroAooopgFFFFABRRRQAUUVJb27zybE+/S2ALe3eeTYn36nuLhII/stt/q/wCOT+/RcTpAn2W2/wBX/HJ/fqpWX8TVgFFFFbAFFFFABRRRQAUUUUAFFFFAHqvwy/5E2P8A6+p//Rlc78Yv+Qlpv/XjP/6Mrovhl/yJsf8A19T/APoyud+MX/IS03/rxn/9GV/DvCv/ACdCt/19r/8Atx6k/wCAfHtv/wAeqf7lfQmj/wDIF0z/AK9YP/RdfPdv/wAeqf7lfQmj/wDIF0z/AK9YP/Rdf2dh/jPPmW6KKK9QzCp7e3e4f/nmiffkk/got7d7h/8Anmiffkk/gqS4uE8vyYf3cH/odYtvZAJPOnl+TF8kH/odVaKK1S5QCiiimAUUUUAXo5Ev08mV/LnT7kn/ALTeqMkbxyOjp5bpRV6ORL9PJlfy50+5J/7TesP4XoMo0USRvHI6OnlulFbiCiiigAooooAKKKKACiirUEKRp51x/qP4I/79S3YBYLdPL864/wBR/BH/AH6guLh7iTe//AI/7lFxcPcSb3/79/3KjrNJ7sAooorYAooooAKKKKACiiigAooooA4/xZ/yFf8AgFZul/8AIRtP+u8f/odeaftHeD/iV4g8dWE/grWbnTtPTS0jnjh1r7KHm8+f+DzP7myvK1+F/wAfFk+XxXqA+nikf/HK/iTjDK6GIzzE1amKjD3z95yPM69DLqNNYacyh+yOpX4geKuf+YTJ/wClVvX1RXyVof7PXxb8L3Ut1o15HpNzIm1prHXI4pHT0yknStr/AIVj8fP+hpv/APwpv/tleNmmEwmY4lV4YqCsrG+W4vFYHDujPDTPpuivjK/vvizpvxAh8HS+LtaOuSzQwxxx66/lb5FR0/eeZs/jHeuzf4Y/HtkGPFN/n/sZsf8AtSvOnw/SocrqYqK5/wATuhns6nPyYaXuFn9spS0Pg36Xg/8ARNfS+h2af2Ppt1c/6j7DBsj/AOe/7iOvlu+/Z3+LHihba68T6h/a9va7/IhvvEMcsjH/AJaRpvk+T/bq1dfDv49XM24eJ7yCOMbI4YfE2xFT+4n7yvQrYPCVMJQwyxUP3fN/5MefRxWKp4mtiPq0vf5f/JT6avLx7yTzH/65pHH/AAVDXxjpV98W9Y8eXXhC38XawNat554Hjk1x0j3why/zl9v8B5zXZr8Mfj338U3/AP4U3/2yvOrcP08PpXxUUehRzqeI/h4aQz9pZC3xp8IHPSztv/SqavqTUP8AkJXf/XeT/wBGV8ka1+z98XNf1KHUNUvY9RvoVRILu81yNnXZ9wI7yflWx/wq/wCPe3H/AAleof8AhUj/AOOV6eMwuExVChT+tQXszz8LisVh61ap9Wn759M1PZ3n2fzEdPMgf78dfG/jyz+Mfw30221DWvFurJb3E3kp5GvvP8+zf/BIa2dA8D/HLxFo9lq1l4q1JrS7hWaEt4l2OU+hkrzv7Bpwh7b61HkZ2f23Oc/Y/Vpc59A/Gyz8j4S+MXR/Mgk09/Lk/wC2kded/slqV+Fd8Cc/8TiT/wBEw1y6fDH433Fnc2Gp63c6tpt2vlXNjdeJQ8c6e/7yqdn8AfjJ4Ns5IdA1qSw0ud/OT7D4hjhR3/20ST79ejTwuEp4Cpg/rUNZXOOpisV9dhivq0/hPqbS/wDkJWn/AF3jr82dXupnup0eZ5E81jsZ6+iLf4W/H+SaMR+Kb/fu+T/iqR1/7+V813SvHM6ynL7vm969jh3AUsJKq6daNT4fhPF4gxtTFRgp0ZQ/xFWiiivsz4sKKKKACiiigCaORoJFeNtjqfvKa+yv2RYpdS+G9/Gu6SR9Wky7v/0xhr470/T59Sukt7dN8j9B6V7l8OPCvxF1TwldQ/D7XLqy0VLwxXc0OrfYftM2xednmDKbNleJnVCniMK6dSfJ5n0GSV6mHxPPThzn2ZZ3CW99aWtn/q/Pj8+4/v8A7z/0CvkD9kqMp8S/EhJ66TL/AOlMFaUfwv8Aj8vzL4r1AH1HikZ/9G1heH/2evi54ZvpLzSL5NLupF2vPY65HE7p/vpJ0r5LB4bCYXD16EsVB+0PrsVisViK9Gv9Wl7h9a0V8xn4YfHzt4pv/wDwpv8A7ZXG6hefFrS/Htt4Nl8XawdbnmghSNNcfy98yI6fvN+3/loO9eVR4ep4i/ssVF2PQnn06H8TDSPsLWP+Rf1r/sHXX/oiSvnv9jBStn40/wCulj/7Xqo3wt+PFxHKsnie/eN02tG/iUHenpjzOlZXh/4C/F/wqJxot+NGFwEEwsddjh37fub9kn5V6NDB4Slgq+E+tQ9/lPPrYrFVMZRxX1afucx9YUV8yL8MPj4FOfFN/n/sZv8A7ZXGaZefFrVPHlz4Qt/F2sDW7eWeCSOTXHSPfCHLjeZNv8B7151Ph6nUUnTxUXyfgehUz6dPk58NI+zq+W/2olJ+LXhI56WNt/6VTVZ/4Vj8fP8Aoab/AP8ACm/+2Vi61+z18W/El/Heapepqt5Gu1J7zXI5XRP995K9HK8LhMurOtPFQd1Y4MyxWKx+HVGGGkfXeqf8hK7/AOu8n/oyoa+Z1+GHx7Yf8jXfn6+KR/8AHK5nx5YfGP4c6TBqWs+LNWSzmn+yp9n8QNN8+zf/AASeleZSyKlWqclLFRbf9dzvqZ1UpU1OphpaH19XzJ4oH/Gaejn/AKftL/8ASaCqeh+B/jj4i0Wx1Wx8U6k1tdRLNDv8S7HMf4yVVm/Z/wDi5N4gj1ye9jl1dZEdNTk1yMzh1+5h/M35r08vwOEy+dTnxUHeEoHn4/G4nHQhyYaXxRkfVlS18yj4Y/Hv/oab/wD8KX/7ZXH/ABBl+Lvwz/s8654t1iJb7zPJNtrrzfcwHzskOPv/AK15tHIKWInyUMVFs9CtnVTDw56mGkfolRRRX+gx/OIUUUUAFFFFABRRRQAUUUUAFFFFABVuC3Ty/OuP9R/BH/fogt08vzrj/UfwR/36guLh7iTe/wD37/uVjrLRAJcTvO+9/wDgEf8AcplFFapW0QBRRRTAKKKKAOH8XeIPGmh/EfwFP4R8I2vie9tdUS709Li1+1fab7y5P3H/AEw/d/vPM/6Z79/7iv1L0Oa8uNJspr62WxvHgRprVZvOWF8Dem/jdg5G7Ffk98YIvF2k3GleJ/DmtXlpFoz+fixfZPZT/wDPf/ppH5f7v/7CSSvdv+HkF7/woD+0f+Efb/hYf2r+yvO+ySf2T5nl7/tXmdOn/LDfv3/7Hz1+F8S0Kjx8mkdmDxEKMp859+7hwc8CvBrf9sD4ZN8Y9V8Av4gSG9sYHkbUriRE01p4/MM8AmPAkjVMv/AMOM743A/NvwX8X/iboej+NE0/xpqFsPFw86/eaTfNNN/y0njf/lg7p+78yP8Ag8v+4mzj/D/huHT4I5LmBPM/gt9n7uCuLB8P4nETjzaRZs8wnPl9nA/Wv4L/ALT3gj493mr2fhe9nEtif3cd5F5L3dtkD7TCueU5HBw6ZTeib03V/wBoDSdESxtNQml+z6+5McMcZ+e4j/j3/wCwnXf26fx1+aXwT8N6rP8AHDwlp/hvxPbeEb6+vd8OoXc+x7Z0OP3Ef/LR3j8xEjf5JP3kbgpxX66eN/A2n+PNJ+x3qbHT54LlPvwv6ivIxuDlga3sZm9GcsXRnCZ8lSSeXHvevaPhv8Eo5tMnv/FMTeZcw+XDYn5PJj/vv/00/wDQKv8Awy+C58PapJqXiAw3l1bTH7FHGuY0/wCm/wDvn/xyvZOntXnl0ML9qZ8keOvAN74B1T7Nc/6RYTf8et//AH/9h/8Abrm6+yvEHh+x8TaTPp+owCe1mGHjPevlzx14B1DwLrCWkySXdtcvssrpE5m/6Z/7/wDsf8tKCK1Hk1RzEcbySRoiPJI7+WkcaeZI8le7eCfgfYw6HOPElubjUL5PLCxSZ+xf7j/3/wDb/wCAeu+58J/hWnhWOHU9YiWTW5U/dr98Wienu+M5f/gHu+r8SPjZ4L+FOseHtO8VeIbXRLvXbr7LZLNwGbkb3/uJnam9/k3vH60y6ND2fvzPDvGHhO58B6tHpl/572Mz/JfQJ+8uU/5abP8App/sV9KeAhoy+D9K/sCQSaN5A+yuGyPLq34g8N6Z4q017DVLZLy0k+YxyVdt7WKzt44YkSOBF2oiLhFWg6KNH2cro/Or9sL9uLV9a8ST+BvhXqd1pdtpd6Y9Q8QWPyT3N1DJ/qIP+mCOn7x/+Wn3Pub9/AeJv2z/AIy63428G6zb39vp0ejQok2m2Rf7DqUmMTvdJ/tj7ifwfwfvKw/2nPFs/ib9qDxeZvBUfhe8t/8AQZFEf+lXKR/cvpv4H3p/c/g2f6yuFr9AybJcNjsN7aoeFUqVJ1J2meueFf26/iN4J+NWpeJfE0s2teFdSdI7nw3E5MdrapnY9jvxiePJ37/9d/H5f7t09R/a5/bs0/VPDn/CH/CzUBdSatZJJqHiOFeLWCZciCBH6zOj/O//ACzBx9//AFfgPg/4bweKI0vtbtfM0lH8yC3k/wCXr/7X/wCh12/jTwTY+LPIuvLhj1a1Ty7W72f8s/8Anm/+x/6BXfPhHmqc9GfuE+0rck4c54do8d1b6bAl5s8xE8v/ALZ1u6H4gvfDepR3Vm//AEzkjk/1c8f/ADzqpeWc+n3UlrcwPbzw/u3jkqCvuKdP2cPZlQPfND1y18QaVBe2e/y3/wCWcn+sR/8AnnV4dKyvC95PeeH7B5tO/suTZ8lvH/q/L/56In8Fao6V7VP4DAWiiitgCiiigAooooAKKKnghed/+eaJ9+ST+Ck3bcAt7d7h/wDnmiffkk/gqS4uE8vyYf3cH/odFxcJ5fkw/u4P/Q6qVkk3qxhRRRWwgooooAKKKKACiiigAq7HIl+nkyvsnT7j/wDsj1SoqZR5gCSN45HR08t0oq9HIl8nkyv5c6fck/8Aab1RkjeOR0dPLdKzjJvRgFFFFbAFFFFACGrdvCiJ51x/q/4I/wC/RGiQQfabz93An/j9cL478eahZTWj6P8AZdQjdJN/lp5+z/nn9yvGzDMaWBpc9VhOfs/fmHivxfe615A8L3v2y7R/38dpH5myP/gf+3Unh/w/Bb+Rqk1q8etTJ5l1JI//AC0k/wBZ+7+5Unh/wna+H3kntnupJJk8t47h/wDtpW3X4hjcwrZjPmq7HyeKxU64UUUV5hwBRRRRqAVn/wBuad/aP9nfbYft2/y/s/8Ay08yqnjDWJ9D0OS6ttnn+fHH+8SvMpPEF1J4g/tfZD9r3xyfc/d0am9Oj7Q9oorA8F6xdeINHkurzZ5iTyR/u08v/lnHW/RqZz/dhRRRRqRqFVtQ/wCQbd/9cJP/AEXVmkkj+0QOj/6t08usakPaU/Znfl9SnhsTTq1O55bG2IYxXqv7PY/4qnW/awT/ANHJXCeIdFg0WSzghd5NyP8A6yuz+BGoWun+IddkvLy2tI2sI40e6mRN/wC+/wBuv5zzDBzwNSpRq/Gf6UPOMNn/AA4sfg/glb/0rlPvPwl8QPDen+GdJt7jWLS3uIbOGN0d/uvsFV/H3jvw9q/hLVrO01e1uLua1dEjjkG5jjFeHWs0M0CTQzJPC/zpJG+9Hqwz21rayXV5PHb20f35JH2J/wB9163+tGMnhvqvJH4eX9D8J/1bw0K3teeW54H8bLFf+E6NzdH9x9hh8uP+OWTe9eZateS3m525+T5I0/grv/jhqkGpePhNbXMN1B9ghj8y1mR0X53+T5K81uJPLhCf89K87L8NPG1aeHon7nUzTDZBkMcxx/wRUf8Ayb3SnHU9nZz6hP5FtA9xJ/zzjqCP93XQ+B/+Rjj/AOuEn/ouv6LjdJI/ziryUpya2udf4Xt57Pw/aQXKSW86eZvjk/66VfvPI+yz+c/lwbJN/wD1z/5aVLXD+NPFmo6PqUmnQpD9nmtfn8xP3nz+ZHVHlwh7SZUuPP0//kQ99wjp/pX2f9/+8/5Z/f8A+B13ukeINO1h/ItL2G8nRPMeOOvIdD8WXvheCf7GkMm/y5P9ITzPuV1l5p7+A4IL3REmvJ739xPHcJ5+yP8A1n8FfT5Hnk8vn7Op8H9fCe5Qr+w9yZ6J0payfD+sJqGm2HnTQ/b3gjknt43/AHiSf8tPkrV6V+1UK8MRD2lM9wsWl55G9HTzIH+/HRcWn2f50fzIH+5JUFT2l55G9HTzIH+/HWjTWqAgoqee1+z/ADo/mQP9ySoK1TTV0AUUUUwCiipLe3eeTYn36WwBb27zybE+/U9xcJBH9ltv9X/HJ/fouJ0gT7Lbf6v+OT+/VSsv4mrAKKKK2AKKKKACiiigAooooAKKKKACiiigD1X4Zf8AImx/9fU//oyud+MX/IS03/rxn/8ARldF8Mv+RNj/AOvqf/0ZXO/GL/kJab/14z/+jK/h3hX/AJOhW/6+1/8A249Sf8A+Pbf/AI9U/wByvoTR/wDkC6Z/16wf+i6+e7f/AI9U/wByvoTR/wDkC6Z/16wf+i6/s7D/ABnnzLdT29u9w/8AzzRPvySfwUW9u9w//PNE+/JJ/BUlxcJ5fkw/u4P/AEOu5t7IzEnnTy/Ji+SD/wBDqrRRWqXKAUUUUwCiiigAooooAKKKKAL0ciX6eTK/lzp9yT/2m9UZI3jkdHTy3Sir0ciX6eTK/lzp9yT/ANpvWH8L0GUaKJI3jkdHTy3SitxBRRRQAUUVaggSNPtNx9z+BP79S3ygLbwJGn2m4/1H8Ef9+oLi4e4k3v8A/sUXFw9xJvf/APYqOs0ne7AKKKK2AKKKKACiiigAooooAKKKKACiiigDj/Fn/IW/7YVj15z+0b8U/Gngfx1Y2fhrQ7fU7CbS455JpLBpiknnzp99P9xK8oj/AGifis7bU8HWp9hpM/8A8XX8QcX5LicTneJqQ5Pi/mP3zJs5w2Hy+jCfP/4CfTtRV8r2H7U3xC1aZ4bHw9pN5OvzssFhM7p/4/V1f2iPimf+ZRtR/wBwqb/4uvkv9XMbT0bh/wCBHsR4gwdT4Of/AMBF8Tbv+GztJ9P7S0v/ANEQV9U2dmnkfarn/UfwR/8APevi4al4y1D4jR/EHU/DkyX9vPBPBappsvkzvAiIibP7n7sb+fX1rsJ/2lPipeTbpPCdlgfIijSZ8L9Pnr3sxymrioUFTcLwhGJ4WX5lSw86zqKfvzPpq8vHvJ97/wDAI4/4KZXyrqX7UXxE0kr9u8O6XYeb9zz9Plj3/wDj/NWf+GivinIpH/CI2pDf9Qqb/wCLrwf9Xcavf9z/AMCPd/t/Bu9P3/8AwEd8P2P/AA2Jrwxx/aGsf+i56+n6+KNK8TeO9G+JN341h8NzSavcz3MzxyafKYN8wcP8v/Azx9K7g/tD/FMNj/hErXH/AGCpv/i69vNspq4ypCdNx+CMdZHjZZmVLBwnCanrL+U+n6K+Vbz9qb4iaZcLbXfh3SbSdjvWGewmR3/DfVw/tFfFeP73g21H10mf/wCLrxP9XMav5P8AwI9n/WDB/wB//wABOq/bAz/wrnQMf9BR/wD0SK9N+FZ3fCvwcfXSIa+X/iL8QPiF8T9FttM1Tw19ngt5/tCfYtMljffs2dD9K3PD/wAbviZ4Z0Gw0q18JQy22nwpbQyT6VN5hROm/wCevbq5TWqZbTwl4c8JfzHj0cypU8xnirT5Jx/lPqup7O8+z+Yjp5kD/fjr5Vuv2kPihZwvPP4UsreBDvdpNLmRE/8AH6ZYftOfErVYmmsvDOm3lur7d9vp0zpu/wC++teJ/q5jfj9z/wACPY/1gwfwe/8A+An1vb2f2fVbB0fzIJJ49kn/AG0r8yNT4vp/95v519MaR+0x8UrS4Td4Ntp4ndC8B0mfD/8Aj1fO+uac1rcC7STz7S4dmSdV2987CMfK3tX2fD2W1cvnW52nz8ux8dxBmFLHRhyIwKKKK+xPjAooooAKt6fp82qXSW1unmSP+lGn6fNql0ltbp5kj/pWpqGoRafbvpunPuRv+Pm6H/Lx/wDYUALqGoRafbyabpz7kbi5uh/y8f8A2FfVP7JP/JKr7/sMSf8AomGvjj1r2b4S/Ffxr4J8MzafoGhxajpzXjzvPJYPP+8KIuzenb5E4rxc6ws8ZgnSg9WfQZHiIYTGKpM+zaK+Yk/aK+K0jbE8HWp/7hM//wAXVCw/am+IerSPFZeH9KvJlTcy29jLI6L/AN99K/Nf9XMa9nD/AMCP0T/WDB/3/wDwE+rK+Y/FX/J6eiHv/aOj/wDpPBUbftEfFPaD/wAIlaj/ALhM5/8AZ64rUvE/j3VPiXbeN5PDU39sQTQTpGmny+RvhRET5P8AtmOK9vKcoq4KdSVRx96Eo/EeRmWZ0sVCEKan7k4y+E+0Klr5eX9or4pqjN/wiFmEVep0qbH/AKHVbTv2oviLqu77D4d0u/8AK+/5GnyybPyevE/1cxurTh/4Eev/AKwYPb3/APwE+qq+XvAJP/DZHiHjj+0NY/8ARc9O/wCGiPinux/wiVrj/sFTf/F1w+leJvHmj/Em68aw+G5pNWuZ7maSOTT5TBvmDh/l9PnPFe3luU1cNTrwqOPvw5fiPGzDMqWInRnDn9yX8p9r0V8vn9oj4pj/AJlG1/8ABVN/8XVK+/am+IWmzJb3mgaVZzsu5UmsJkd//H68P/VrGVNE4f8AgR7MuIMHD4+f/wABPq2vEf2uGK/DDR8DP/E3/wDaL1yDftEfFaOT5vB1r/4KZ/8A4uuV+InxE+IXxL0WDTNU8M+RBFc/alaz02VHL7dmP/Hv5V6eVZLiMLi4V6rhZf3jz8wzaliMLOjTU/8AwE+nfhd/yS3wh2/4lsP866qvlTQPjh8TfDeiWOl2vhKGW20+FbaKS40qbzCif3/npzftRfERdVjsH8PaTHfvtRLX7BMH+b0TfXNieH8ZWxFSrTcNf7xvQzzDUqEKdTn/APAT6pr5r/bK5tvB4xwWvMn/AL81T/4aI+Kef+RRtf8AwUzf/F1yvj/xB49+MkunpqnhySyg07zP31npsyoivs3F/v8A9yvRyfKMRgcbDEVXC3kzjzbNqWMwc6FNT/8AAT9EqKKK/vI/AAooooAKKKKACiiigAooooAKtW8CRp51x/qP4I/79Lb26Rx/arj/AFH8Ef8AfqC4uHnk3v8A/sVhf2miGJcTvcSb3/79/wBymUUVslbQQUUUUwCiiigAooooAI/3dUf7D07+x/7I/s61/snZ5f2DZ+48v/rnV6k21jUp06nxoDxnxp4Pn8L3fnpvuNNmf5Lj+5/0zkrm6+iLi3gvIJILlEuIJk8t45P46yP+EL0iTw/HpD2v+iI/mJ8/7zzP+em+uSeG/kNOc+cfEnh/7ZHJdQ/69E+ePf8A3P8AnnX6sfsdal8RtY+CGj3XxJRTqcx3WM8jf6XcWJQeS90vaY8n12FN/wC831+cPxt1G5ute0bwrpHh6YX1zGkFtJaQfPe7/uWkCJX3F8J7yb9h/wDZlin+KPiabUbncZLLQ4WWT7LI6jy9NtP45OhP9xPnxsjTNfjHEnsfrPJR+P7R04Lkp1Z1D2j45fHLw38AvA8/iLxFOxB/c2ljAf395NziOMZ//Z5r89PCP7dnxZj8feJ/Ehez1Ox1O1ZIdDvHf7DpsgTMBh/j+Q53/wDPfP8AB8mzzDx18XvE/wAffH1/4g8WP5lm/wC4TTLd/wBxp8H8CQf9NP8App/y0/8AHKiuNHTS4I/s37ywf/USR/5+/XRk2RfWv3lf4CKmKqVp81P4D6G/ZF/bi1fQfEw8G/FfVrrVbLUrzbZeI7z/AF1ldTyf6ifZ/wAsHd/3b/8ALHeIz8gHl/pBJEjdRuI55r8cfg18SrL4U/HTQPEeoeFYfEVtZb43Mn+vgR/+XuD+Den7z7//AD0k/wBXX62+CfGGi+PPDem+IPDupW+q6Pfwie2u7ZyUdTn/ADt/h6V85mGFjhMVOEDvwFb2kJRnI89/aO/aG8P/ALOPgp9Z1K2mvNVupHtdH0qH5JL64KFym8jCJxueTsBwHfCH8nfGXiLxP8bfE+seKvE10uoareDD5UpAkf8AyzggQ/cSP/P7ySSvvT/goR+0P4U0TwZqPwuh0+z8TeKNShSS6jvMPBo0f30nk/6bd44/+ByfJ9/4b0uR7jTYHeD7P8n+rr2+H8vo4yU/bHn42ftq3sVM+n/2Lv20G8Lyaf8ADn4jajINLDx2mka7dSYezf8AgtLp/wC4eNkh/wBx6/Rrr7ivxCuPBc/jCf7LYQfaL90/1cn+reP/AKaV9M/sc/to3vgG+s/hr8TLuZNJST7Lpus6lIRLpcn/AD63Uj9Y/wC4/wDyz+4/yDenBm2UzwM/I0weM9j+5rHA/tcw/EOH9pDWT40uLOaCSDzNJFr/AKn+y97+RsT76Sb9+/f/AB7/APln5deX12/7T3hnRdD/AGoPFbab41uPFF1dP9rvPtI33FndSf6yxkn+46ImzZGnMabE/wCWf7ziK/Q+HP8AcDnf8SZ7d4Lj1SPw7af2rOlxPs8yD/nokH/LPe9bi1zfw7s4LPwrB9mvX1COb95/0zgk/wCWkez+CukWvvqPwGZh+JPB+neKPIe53xyQP/rLf/WPH/zzqOTwHokmsQaj9iSPyU/494/9Q8n/ACzk8uugwKMCnyQELRRRWoBRRRQAUUUUAFFFTwQvO/8AzzRPvySfwUm7bgFvbvcP/wA80T78kn8FSXFwnl+TD+7g/wDQ6Li4Ty/Jh/dwf+h1UrJJvVjCiiithBRRRQAUUUUAFFFFABRRRQAUUUUAFXY5Ev08mV9k6fcf/wBkeqVFTKPMASRvHI6OnlulFXo5Evk8mV/LnT7kn/sj1RkjeOR0dPLdKzjJ7MBAMVct7dI4/OuP9X/BH/fpILdI0+03H3P4I/79cL4w8Wf25qN/4X8h47u68u0S73/uE/5aV5WZ5jTwNH2siKk4U4c8w8deOprHWP7HltYZBewJH5nn7NnmeZH9yrHhPwmnhOO7RLr7R53l/wDLDy/ueZR4T8Pv4f02S1uXhuJHn8zzI0/65/8AxutuvxLH4+rmFX2stj5bFYr24UUUV5mpwahRRRRqGoU/7O/9x6SP78deBUam9Gj7Q9V+JH/Iqyf9fUFeVV6j/Yb+IPAGm2UM6W8myCTzJK4WTw3PH4m/sTz4fP3pH9o/5Z/6vzKyOyj+79w7f4X/APIuz/8AX9J/6LSutrE8J+H5/DelSWs06XG+fz/Mj/7Z/wDxututdTgn8YUUUUameoUUUUaiMvWPD/8Abk9o/wBq+z+Skkf3PM+/XCYMgIbivUY/3clcZceF5PD9rb3upTpJBM/lpHb7/MeSvzPizLPgxVCH+P8A8l5T+tfBnixRjUyXHVv5fZR5f8UpfZ/9KPo34Z2qxfDnwy83KHTYTs/vfJVX4tN53w38Ubv+fP7n/AkrgND+PFlpfh/TdPk0S+Z7O2SHfDJCifJ/c5qp4u+Nln4m8KaxpUekX0M15D5KTPIjonz/AFr8k9nP2tz9hp5TmH1xVp09Ofy/mPJJJfs6uSmT1qg0wmGAMCrN5J+72VU5r9e4Vyz2cPrNeHv/AGT8e8XuK5YicMmwVb3P+Xsf70ZRlH7P/pIc10PgP954mjRP9Z5En7v/ALZ1Q0PQ38QTzpDOkfkp5n7yuh8aWf8AZ/gCCB9kkkLwI8kdfpOp/LU9fcOz8t4/4Hryv4kf8jNH/wBesf8A6MkrN8H/API3aL/19V1/jDwfPrF1PqiXUMccNr/q5N/mfJ5klZBTh7CZ5tJ/x7yf9c698t/9TH/1zjryDw/4Tn8UQXfk3UNv5Plx/wCkb/8AlpXsEcflxxpRAMTUOI1TS/8AhC9Su/FHnpeSPPJH9nkTy/8AX/7ddf4X1t/Evh+DUHgS387zP3cb+Z9yTZVjy/M+/XH6x4fn0vxBP4o8+GS0tXjn+yR/6x9kez/cr6zJs6nlc/Zv4P6946MFjeT3Jneq2aWsnwv4kj8UabJdQwPbxpP5HlyVqq26v2jD4iGKgqtJ+6fSb6li0vPI3o6eZA/346Li0+z/ADo/mQP9ySoKntLzyN6OnmQP9+OtmmtUBBRU89p9n+dH8yB/uSUyCB55NiffrTmVrgFvbvPJsT79T3FwkEf2W2/1f8cn9+i4nSBPstt/q/45P79VKz/iasAooorYAooooAKKKKACiiigAooooAKKKKACiiigD1X4Zf8AImx/9fU//oyud+MX/IS03/rxn/8ARldF8Mv+RNj/AOvqf/0ZXO/GL/kJab/14z/+jK/h3hX/AJOhW/6+1/8A249Sf8E+Pbf/AI9U/wByvovQYHn0qw/5ZolrBveT+D93Xz5o+nyahBGifu40TzHkk/1aJX0PZ3Cf2HpsNv8A6hLWD/gf7uv7MoX59Dz5lqedPL8mL5IP/Q6q0UV6iSWxmFFFFUAUUUUAFFFFABRRRQAUUUUAFFFFAF6ORL9PJlfy50+5J/7TeqMkbxyOjp5bpRV6ORL9PJlfy50+5J/7TesP4XoMo0USRvHI6OnlulWoIEjT7Tcfc/gT+/WrkkriC3t0jj+03H3P4I/79Q3Fw9w+96Li4e4fe9R1CT3YBRRRWoBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHJ+KLiePVfknkj+T/lm9UdL1C6/tK0/wBKm/18f8f/AE0rgPjt8dNI+FvjC00zUdJvL97rTo7tJLWZET/WSJ/7Trz20/bI8L215DN/wjOrfu5Ef/j6h7f9s6/hvjDJ8bXz3F1adLTnP6ByTNMFRy6lTqVTjv2QneL4ieKXSRo5P7Jk/j/6erevrWzknjg+1Xl1N5H8Ee//AF1fEnwH+IWn/CnXtX8Sa1p9ze6df2j21vBbSKjSv58b8/7H7vmvV7z9s/w9eTl38O6r/wBM0+1Q/J/5Drgz3KsVicWq1GHMkjPJ8ywtDCOjUnynvt5rF7eT73un/wCucb/cqP8AtC6/5+pv++6+fv8AhsPwv/0LWrf+BUP/AMbo/wCGxPC//Qtar/4FQ/8Axuvmv7CzL/n1+J9B/beXf8/TJ/bMuHnj8El3eTAvMeY2f+eFfSGn3l1Hpem/6VN/x4wfx/8ATCOvjD49/GDTPir/AGD/AGbp13YiwMxf7VMj79+z+6P9ivU7f9r7w3Z2drbyeHNWkeCCODP2qH+CPZ/zzr3sVlGOqZdhqMI+/Dmv/wCBHi4XNcHTzDE1py9yfL/6SfQf9oXX/P1N/wB90f2hdf8AP1N/33Xz9/w2J4W/6FrVf/AqH/43R/w2J4W/6FrVf/AqH/43Xg/2DmX/AD5/E9xZ3l3Sqc1+01NJN8cPB8jyPI/2K2+d2/6epq+rtQvLr+0rv/Spv9fJ/H/00r4a+LHxcsPiB8QtE1+zsLi0tNPghgaC4kV5G2TPJn8d9ew3H7Yvhi4uJ2/4RnVcyO8n/H1D3/7Z19BmWU42rhsLTpQ1hHU8LL80wdPE4mdSXxyPfP7Quv8An6m/77o/tC6/5+pv++6+fv8AhsTwt/0LWq/+BUP/AMbo/wCGxPC//Qtat/4FQ/8Axuvn/wCwcy/59fie5/beXf8AP09Q+Mt5PJ8IPGW+aaT/AIl/d/8AppHXB/sjXM0PwrvSk0kX/E5k+42P+WMNcj45/ai8P+K/AuuaRbeH9St5tQtvIR5LtHRPnz/crnvgf8fdI+GPg+50fUNMvr6aS/e6ElrMiJsKIuPmH+xX0OHyrG08pqYdw99yPDqZpgv7UhX5vc5T7E0u8upNStP9Km/18f8AH/00r83bbVhZXl7Dcp9rspnxND+P3154evqKx/bG8MQ3kM3/AAjOqgxukn/H1D2/7Z18j3knm3Esg+67b8fjXr8M5fi8DKs8RDlvy/qeTxFjsLioUfYS5viLesaP9i2TwP8AaLKbmGYfj8j8cPWTWvo+sfYt8E6faLGbiaE/h86c8PSaxo/2LZPA/wBospuYZh+PyPxw9fcHwZk1b0/T5tUuktrdPMkf9KNP0+bVLpLa3TzJH/StTUNQi0+3fTdOfcjf8fN0P+Xj/wCwoANQ1CLT7d9N059yN/x83Q/5eP8A7CsCiigB3qB0r7N/ZDuJ7f4V32yZ4v8Aicyf6tsf8sYa+M+c9Oa99+CPx+0j4Z+ErjRtQ0u+vZ5b5roSWsyImwoiY+Yf7FeDn2HrYnAyp0Y8zPpMhxFHDYxTrS5UfYel3l1JqNp/pU3+vj/j/wCmlfI/7IskkXxK8TMjPG/9lTcxtz/x8wV29n+2P4Yt7yGX/hGdVzG6P/x9Q/8AxuvFvgn8VbD4Y+KdW1S/sbi/truza1SO3kVHXdMj5/8AHP1r5HLMpxmHw2KhVjrNRsfV5hmmDqYnDVKctIcx9xf2hdf8/U3/AH3R/aF1/wA/U3/fdfP3/DYnhf8A6FrVf/AqH/43R/w2J4X/AOha1b/wKh/+N183/YOZf8+fxPf/ALby7/n6e563eTyaBrO+ab/kHXX8f/TCSvnr9iy4kgtfGxSR4/nsf9W2P+e9Xr79rjw3fabfWyeHdViee1ngST7VDxvjdP8Ann715l8AvjNpfwpg8QR6lp11fDUDblPssyIU2eZ13D/br38LlONp5biaLh78+X/0o8TFZrg6mYYatCXuQ5j7S/tC6/5+pv8Avuj+0Lr/AJ+pv++6+fv+GxPC3/Qtar/4FQ//ABuj/hsTwt/0LWq/+BUP/wAbrwP7BzL/AJ9fie3/AG3l3/P0+gf7Quv+fqb/AL7r5X/amkkk+L3hNnZpH+wW3Mjc/wDH1NXUL+2F4Xb/AJlnVv8AwKh/+N15B8Yvi1p/xE8baPrVjYXFpbWNtDA0Nw6u7bJncn8d9fR5DlONweIlPER3TPBznMsFisKoUJH3PqF5df2ld/6VN/r5P4/+mlV/7Quv+fqb/vuvA7j9sbwvcXU8n/CM6r+8d3/4+oe//bOov+GxPC3/AELWq/8AgVD/APG6+eWRZlzfwvxPcjneXqP8U+gf7Quv+fqb/vuvmHxTJJJ+2vo7u7b/ALdpeZN3/TtBW8v7YXhdunhnVf8AwKh/+N15jcfES28TftAW3jm20+4h0u1uLW5mtpp181Y4Y40f5/8AgH619Fk+U43CyrurDeEjw82zTB4iFH2c9pxPtLT5L28k8tLp444/vySP9yp7zXJ/L+y2080cCf7f7x/+mleA3n7ZnhbyPstt4c1iO0T/AJ6XMXmSH+8/HWqP/DYnhb/oWtW/8Cof/jdfP/2HmX/Pr8T2/wC28v8A+fp9h0UUV/oofzIFFFFABRRRQAUUUUAFXtPsPtHzzf6v+CP/AJ7Uy3t0jj+03P3P4I/79QXFw88m9/8A9isHepogC4uHuJN7/wDfv+5UdXv+Ql/1+/8Ao7/7OqNOm+gBRRRWwBRRRQAUUUUAFFFFABRRRQAUUUUAeb+O/iJ46+FPjzw34t0CWGHS9Mn8yD9xvjeR/knguv8AfT5P3f8A6MrA8WeJvGP7XXxMu/EPiC5+x6Ra/uILe3+e30uD/nhB/wA9J5P+Wkn/ALJsSvX7yzgv7Se1uYUuLSZPLngkT5HSmaTo9loOkwadpsKW9japsSNK+FxPDscVj/rU5+6RyT/n9w4jxR8M7GPS0fw9apZyWUHl/ZN/+vjT/lp/v/8AodcRocjyT/Zdn2iCb78f/tT/AGK7D4oa5qPn/wBkeQ9nYOm95P8An6/+wrntD0+fxBJHpejp5e9PMuryT+D/AOw/6Z/x17sqcF+7pnXDQxfHkdrpfhyRLb/SLSb92l/H/HP/AMs4/wDY/wCudfo3+xNrXw21D4L2Vp8NLeSxtbVwNZsL6RH1CO9bmSS6cZ8x5P74+QoAE2BNg+WdL0HS9P8ADs+gzWSahpN1H5d1Bcf8t/8AppJ/t/8AoFeHeLPCfij4H65dT+FdX1OPSfEEMmlR3emv+/mST/l0n2f8tP7n/jlfn/EmV1pOFaPwmcMT7CfteQ7r9r7WvhdrX7Ql9deCYZJLiSR0165tHj+wXGo7x88af3/4Hk+48hT+Pe9edVUt/Ad14LnSHUrX7Pf7Pkjj/wBWkf8A0zq3X0GRYKeFw3JWHHmqP2kzr/hpb6u+ueZYP9nsE+S+kk/1bx/88/8Afre+Knwsg8c2sl9Z7LfXYU8tJJP9XdR/885P/ZJK0vhneXVx4YSOayS3tIX8uC4j/d/av+mmz/2p/HXVhcV9NUwVLF4f2NYyqQ9p7h4BofwX1TS/Dket3MCfa/8AWf2Ts/eQx/8APT/f/wCmdVP9ZX0VzXG658M7LVNcjvYZ/scDv5l9bx/x/wDXP+5WNDLYYGj7OgOjCFOHIc98K9P1STUpL22n+z6T/q5/M/1c7/8APNP9v/ppXqf8VMt7eCzt44LZEt4IU8tI4/8AVpT1r16MOSIC0UUVsIKKKKACiiigAooqeCF53/55on35JP4KTdtwC3t3uH/55on35JP4Knvv9H2Qw/8AHr99JP8Ant/t0y4uE8vyYf3cH/odFvcJs8mb95A//jlYO794ZUqSOP7RJGif6x6fcW728n/PRP4JP79Lpf8AyEbT/fjpyf7ttAcn8DfiFon7R3iDUdE8CyXNxf2Fr9umGpw/ZU2eYkfyP+8/jdK43xZ+078P/Avi3W/DesXOrRato99Ppt1HDp4dPPgkdH2P5nzjeleT/wDBM/40eCPgj8UfFureOfEUPhyxvNFW1gnngmk3v9rgcp+7jf8AgQ9q+eP2gPEGn+LPjx8SNZ0m5F/pWpeJNSvbS5T7ssL3Ujxv+KGvxGPF+ae3cPd5T6L6hR5D7u+G/wC0J4M+LfjrRvB/hmfUrjXNZn8i1jurMQQb9m/538z/AGK+hf8AhQ3jORP+PbTP/Bj/APa6/LP9jfxnonw7/aY8A+JPEmoR6Toemah511fSK7pCvluOic96/WZf29/2e9n/ACVLTf8AwXX3/wAYrzMZxpnNGf7iEf8AwE6cPluGn8czwr46eNtL/ZtvtGsfHbzQXGrwPPa/2Yn2tNiSeW+/7leYf8NmfCr/AKCOtf8AgsH/AMcrj/8Agpj8cvAPxr8Q+ALvwJ4mt/EcOm2N1DdPbwTR+S7z70/1kadq+Ic+9elQ4wzWVPnny39DnngKEJ+4frf4y1CD4d/B7RvifrW+Pwhq0FlcWslp+/u/Luo/Mg3wfwf9/Ko+DfFumePfC9j4g0V5pNKvd/kSXCbH+R3T7n++leZftFftIfDLxt+wN4J8BaL4tt73xdY6b4ehutJSCffHJBb7JxvePZ8ns9Xf2Vv+TefBn/XO6/8AS6evpOF8+x2aYmdPEnJjMNRoQvA9Zooor9SPGCiiigAooooAK1bONLiGN7v7/wDyx3v/AK6qlvCiJ51x9z+CP+/XBeLPGF74gjg/4RW9e8u438yeO3RPkT/ln9//AG68PM8wo4GnzVWFSpCnDnmP8V+I38TzXeiaZNdW+uwz/P8A8sI0jT/WfvKseG/D/wBjsbR9StYZNaR5JHu5P3kn+s/d/vP9yr/h/Q7W4jtL28g+z+IfI+e7kf78j/6yOT+Cr0kbxySI6eXIlfieYZhWzGtzS2Pk8Vip1xtFFFeZqcGoUUUUahqFFFFGoajo/vx14FXvdYH/AAgeg/8AQO/8jv8A/HKNTejP2Zb8L/8AIs6T/wBesf8A6Lqx/Zdl9u+1fYrX7Xv8zz9n7z/v5U9nZwafawWsKeXBCnlpHUtGpmFFFFGpGoUUUUahqZvibxFZ+EvDeo65qjzR6dYQ+dN5Cb32eZHH9z/tpXRfAvQb39pPwjfeJvh+iXej2eoPpc0mrP8AZXMyRo/3P3nybJ0rzH9oD/khPjz/ALBkf/pRBWl/wT78da/8PP2KfiRq2kag9hLa6hrmpQfuUffNBpNvJG/zxv8AxxisZzPqcpyyhjqE6kz6ItP2WvHwkDzWOk+Wv/LOPUD8/wD5DqDUv2V/iFrTRm70rQ7jy/uRy3hdE/8AIdeTfsG/tffF342ab4/uPGPjGTVn0qbTY7QDTbKAJ5/2rzPuQpn/AFcfWvuD4S+MtW8Vatf2+qXn2lIYEkT5I0/j/wBis3L2isz6Ohk9LDVPaUpygz49vv2P/imuoXXlaDpog3vsKamgGz/v3ULfsd/Feb5Bommp/wBNH1ZCP/RdfY/hnxhrWpfFDUdHubzztNV7rbb+Qo2BHwnz9eleSftffGTxt8L/AIc/EzVPC+tNot/olhBPYOLWCUQu8lqHz5kb7+JJOv8ASvlv9XcF7b25+1y8Qs3ngvqXLDk5eX7XN/L/ADHh/wDwxJ8Xf+gdof8A4N//ALRSf8MV/Fz/AKB2gf8Ag2P/AMbqt+zX+1h8WviJ+yl8RfGHiDxjLeeI9MbWjY340+zi8n7PpUc8A8tINh/ePu98V8af8PP/ANpc9fiX/wCUHTP/AJGr6mnamtD8YqZTSrVPaVJyufUXxH8M6h+ybZ2us/EuOG0sdWd7G1OjP9tcyJ+8+dD5YT93Wd4u1a28RfDex1ax3/YdQ+y3UP2hNkmx6T/gqb4q1fxJ8Ifhz/al6bx/7Ykk8x4Y4/8Alxgf+D3kej4daXa6x8GfBFrep9og/sayfy/M2fwVpCftGfO5tgaGBjCpA5Dwf/yN2jf9fUdexyRpJHIjp5kbp5bx1i2fg/RNPuoLq2svLnhfzEk896261gfMVqntCvZ6XZaf5n2O1hs9/wB/7Onl76sUUU9TDUKiuLeC8gkgmRLi3f78cifu3qWigNTitc8P6pb6r5+j7NP0mFI5J47efyN+yT95+7/3K6zQ/FGneKPPew87y4fL3+ZB5f36sSRpIkiP/q3Ty5K4jXND1Tw/5cfg+1mjSZP3/wBn/efvP+Wf36+kyXOZ5XO1T4D18FjeT3JnonAo+lZOkeItK1iTybO9S4uETzHjjrWHFfteHxFLEQ56MuY+kLmlyP5nk7PMjf78dT6hGlnBstn8yB/+W/8AfqCS4SODyYf4/vyf36jtLzyN6OnmQP8Afjp8rvzgQUVPPafZ/nR/Mgf7klQV1Jpq6AKKKKYBRRRQAUUUUAFFFFABRRRQBXvLxLOHe/Suv8J/DLxD440G01vSobKSxunkjTz7ry5Pkk8uT5P+AUvwv8J6T408aQaXrll/aFi9rPP5HnvH88fl+X9yuvuP+Eu+FPxAS0sIrzSPg5ptyk00myOeCKB498/zvvn/ANe9fhPF3F2ZZDmSw2G5eTlPtsryvDZhhuefx/1/5MY3/Ch/Gv8Az66Z/wCDH/7XR/wofxr/AM+ul/8Agx/+11794X8YaF4y019Q0PUYdTtEn8h5oEeP5/7nz/76Vr/LXyq8Rc5n/L/4D/8AbHZ/YOF/vHkvgv4b69ofhuOxvILXz0nd/wB3deZH88lZvjz4P+JPFF9YPYQWXlw2ro/2i68v/lpXuUcfmUkkn8Cfcr82y+X1DPJ8QUf405Sl/d974v65jb+xqPJyHwbqH7I/xTjtUsbPSNJ+yR/9RuDzJ3/56SV6jpf7P/jW30qwgmtdL8xII43/AOJj/wAtPL/6519QxyH7jn5K8T/aM/aZ8D/BHSNY0XVvGlv4b8bT6FdX2kQyWk8js7pOkDp+5dP9fH/H/cr9Ko8eZzKf7mEP/AZf/JHFUybDU/jmfO/x28X6d+ze+jL46ea3fWUney/slPtX+odEff8A6vZ9+tiCZJ7e3nT/AFcyRzp/uSR76/N74y/tHfET9oKfSZPiB4jbX30lZUs82UFv5IkxvH7lE/uJ19K/RrR+dE0n/sHWv/oiOv17hjOMbmnP9d5fd/lPmcZRhR+At0UUV+gnlBRRRQAUUUUAFFFFABRRRQAUUUUAFFFWoIUgT7Tcfc/gj/v1MnZAW7REngje7/1n/LHzH+/VG7kmkmfzv9Z/zz/uUy4uHuH3vVqORL9PJlfy50+5J/7Teuazpu7GUaKJI3jkdHTy3SiusQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAed/ELwj4f8Q62lzq2h6bqU6WsaJPdWu90TzJPkrK0v4QeDbnE0/g3Q/LAxHb/AGJPMmetH4jePvCXgrXIE8Ua7baa7Wsc8No8c/mTR+ZJ/wA84/ufJXE3H7Qvw+u5vMk8Y2Y/ubLW6+T/AMh1/DXF0cf/AG9iZ0oTtzf3j+g8jlgf7NowqTh8P901rr4f+D7m43yeD9DX+Dyxp6fJUH/CtfBv/QoaJ/4BpTY/jt8OvEHyf8JjZ/2l1/1F1+//APIf36yv+F7fDpOnjGy/8Bbr/wCN18ZOnmvSM/8AyY9unVy7+eH/AJKeH+IvD+kRftbafoiaVaroz6lpyPpqw/uNjxQ+YNn4mvoaP4Z+DDH/AMifoZ/7ckr5v8QeNdAm/aisPEcWqwv4fj1HT5pL/ZJs2RpDv42b/wCA9q97/wCF8fDzGf8AhMLPH/Xrdf8Axuvoc1jjvZ4X2UZ35I3tzfieJlksEp4n2vJ8Uv5TW/4Vp4N/6FDRP/ANKP8AhWvg3/oUNE/8A0rI/wCF9fDr/ocbL/wFuv8A43R/wvr4d/8AQ42f/gLdf/G6+d9lml/hn/5Me97XLrfHD/yU8O8C+G9FvP2qNb0WfR7ObSY77VI47GSL9zGscc5T5PRNg/Kvof8A4Vn4M/6E/RP/AADSvnLwb410DTf2mtY8RT6rFDoNxe6pLHf7JPLZJI5wn8G/neO1e7/8L5+Hf/Q42X/gLdf/ABuvoc8jjvbU/Yxn8EduY8HJ5YL2U/ayj8Uv5TX/AOFaeDf+hQ0T/wAA0o/4Vr4N/wChQ0T/AMA0rI/4X18Ov+hxsv8AwFuv/jdH/C+vh3/0ONn/AOAt1/8AG6+d9lmv8s//ACY9/wBrlv8APD/yU8z/AGpfB/h3w34F0e60rQtN0yeTUpI2ks7fy3dNmRn2r0H4Z/D3wlf/AA28LXVz4V0e5uJtMWSWaazSR2f++9eYftKfEjwn408E6PY6Jrtvqd1DqEk7pDFOmxNmP+WiCu9+Hfxl8DaR8O/DVheeKbS1u7XT1gmhkgnOx/wSvoa8cd/ZVG0Z+05v71z5+hPBf2nWvycnL/dO0/4Vr4N6f8Ihon/gGlH/AArTwb/0KGif+AaVkf8AC+vh3/0ONn/4C3X/AMbo/wCF9fDv/ocbL/wFuv8A43Xz3ss1/ln/AOTHv+1y3+eH/kp0Wm/C/wAEtqVqjeDNEEckyA4skr89L7m8nwu1Q7cV946b8fPhxFfQOfGFmUjdD/x63X/xuvg6+ZWvJ2VtyM7EfnX23C8MXCVb6zz/AGbc/wAz4niWeFnCj9X5PtfAUa6LwvcTNdfYY4PtlrcEGaBn2D/f3/wbP79c7W3NrMVrpgstOWSJZU/0md/vy/7HH8FfeHwhqeJFTRtOS20uTz9OuRl9SA/4+uB8h/ubP7lchWto+sfYt8E6faLKbiaE/h86c8PRrGj/AGLZPA/2iym5hmH4/I/HD0AZNFFFADq+tP2YPCGga78Nbu51XQdO1O6TVnRJ7q3WR9nkw/IPzP5mvkxVPSvqP9mr4neFvCHgC8sdc123067bU3nSCSCeT5PJT5/3aHuleBnsa88FJYf4vI+jyGVCGMX1jY9o/wCFa+Df+hQ0T/wDSj/hWngz/oT9E/8AANKyP+F9fDv/AKHGz/8AAW6/+N0f8L6+HX/Q42X/AIC3X/xuvyz2Wa/yz/8AJj9O9rlv88P/ACU1v+FZ+DP+hP0P/wAAkr588UeG9Gt/2tdJ0SLSLRdGkvdLSTT0h/cMkkMG/wCT/gZr27/hfHw86/8ACYWf/gLdf/G68I8SeONAuv2o9M8SxapC/h+G906SS/2SbNkcMIfjZv8A4D2r6LJI4/nre2jP4Jb83keDm08FyUfZcnxx/lPo2P4a+Ddn/IoaH/4BpR/wrTwb/wBChon/AIBpWR/wvr4d/wDQ42f/AIC3X/xuj/hfXw6PTxjZf+At1/8AG6+d9lmn8s//ACY9/wBrl388P/JTX/4Vp4M/6E/RP/ANK+dvBPh3R7r9qzW9In0i0m0mO+1WOOzkh/cIkcc5T5P+ACvcv+F8/Dv/AKHGy/8AAW6/+N14R4N8a6Bp/wC0zrHiK41WKHQbi91SWO/2SeWySRzhP4N/O8dq+jyqOO9hivbRnfk0vzfgfP5lLBe2w3spR+LX4T6O/wCFa+Df+hQ0T/wDSj/hWvg3/oUNE/8AANKyP+F9fDv/AKHGz/8AAW6/+N0f8L6+Hf8A0ONl/wCAt1/8br5z2Wafyz/8mPoPa5d/PD/yU1/+FaeDP+hP0T/wDSvIv2nvB/h7w/8AD/TLrStB07Tp5NT8trizt/Kdk8l/kPtXo3/C+vh3/wBDjZf+At1/8brzn4+eNvDHxH8E6bYaBr9vqE9vqBuZ9sU6eVDsf533pXrZPHMvrsPbQnbz5jy8zlgfqc/Zzhf/ALdPRfhX8MPC+sfD/wAMXM/hbR5ZpdNinubqe0Q45++5rrtY+GfgyzWO3h8GaH9m+/HJ9iT99/t1wfgj46eA9J+HPhnSpvEtrZvZ2EMc8DQT7/M77yiVvWf7Qnw4EZtrrxhZvZscH/RLrfGf76fu65sVTzL6xU9nGdv+3jfD1Mu9hT9pOH/kpP8A8Kz8Gc/8Ufon/gEleBftYeGtG8Ox+EhpGi6fpJmF0ZfsMRj34MeN/vXuWofGr4fafIkb+MNPkRk3o8cF1sdP+/deB/tPeOPDPjiPwuNB1iHVTbfavOMEUiBN5j2ffjT0rvyKnmEcwh7eM+Tz5v5fM4M5lgZ4CfsJQ5/+3f5j73ooor+/j+eQooooAKKKKACrdvbpGn2q5+5/BH/fot7dI4/tNz9z+CP+/UFxcPcPvesf4miALi4e4fe9R0UVqtNEAVe/5CX3/wDj6/56f89v/s6o0Vm43AP9XRV6P/iZR7H/AOPr+CT+/VGSPy5Nj0KV9ACiiitQCiiigAooooAKKKKACiijzKAJLe3e4mjhhTzJHfYldHd+Cfsel3c32zfdQx73SNKxvD92mn63a3L/AOrR/nrpNQj1DS7rWdllNqEGpp8klv8AwV42Kr1IVFCD0OiEDz7WNHtfEGmvZXieZG/3P+eiSf8APRKXw/odr4b02Oys0/d/6x5JP9ZNJ/z0euj1Tw6+iafazXMyfa5v+WFZVd9Pkqe/AwCp7W88jejJvgf78dQUV0OKkrMRX8T+GLLXNN+y3Pz2j/vILiP/AFkEledeH/hfP/bE/wDbGz7Bav8AJ5b/APH1/wDG0r02kOa5/YwkMX/V/In+rooorqEFFFFABRRRQAUUUUAFFFFABRRU8ELzv/zzRPvySfwUm7bgFvbvcP8A880T78kn8FSXFwnl+TD+7g/9DouLhPL8mH93B/6HVSskm9WMKKKK2EW7e4TZ5M37yB//AByo/s72F1Hs/ef8tEk/v1BVjS/+Qjaf78dYSXKmxnwD+0P8J/ht8PfC+i3ngnxN/bd9dXkkFyn9tWl95cezOdkCDZ+NdH4V+B/wl1Lwnol9f+K/s2oXVhBNcwf8JBZR7JnTLgxum9O3XNY/7GP7K1n+1h428QaDeeJJ/DSaXp329JobFLrzP38cezmRP+eleR/Fjwavw1+KXjLwlBdvfw6Fq95paXTJsMwgmePft7Z2V/LGb/8AChWaw8/Zf4T77AVoYX360Oc+k9K/Zo+FHiDUIbDSdevNX1K4fy4bHTddsrqeZ/SONE3vXSf8MJ6H/wBC54+/78P/APIlS/AH9nY/B/8AZ68N/taWmtyalf6G82onwvNY7IJtl89js+1JJvQ4O/7ldR/w+M11Pk/4Vjpn/g6uq+VqYDHJ/ucTKXrY96GZ4P7eFiclH+wjo8j7E8OePt//AFwf/wCRK45P2ffgt38Ygf8Ac0ad/wDEV7FD/wAFkPEMUiunww0ncvzf8hm6ryH9sr9iKz/ZZ8DeE9bg8X3PiCTWrl7f7PPpiWvljyEkzvSR9/36ulgMbH3auJl+ATzPBvSGFieH6F4P8NX3xf1bQr3Uvs/h22mvEguvtsKb0Tf5f74/Jzge1fUnw/8AHlp4L8N6X4T8Pa9o15a2ZkS2SS6gnuHMkkkn/LOT5/v/APPOvnT9lL4F237RfxgtPBV5rcvh+Cezurs30Nr9oZPJgeT7m9PT1rqvG3wZg/Z+/bCHgG21WTW4tGurUi+mg8hpvMtY5/8AV7nx/rMV+rcP5rDA1oYXk9+X2j4vFUfae8fR9n8TNekvrRHey8t5443/AHH/AE0/66V63Xzxp8if2lYfP/y9Qf8AoyvoZq/csNU9ofNTFooortMwqeCS2t45Jrh/3afwVXIzXE+NPGD6Xqv9ifZYZI72COP7RI/7xN/mR/cryMwxtHA0fa1TOpU9n74/x1421Oxurd9DeHUI3SSR/Lg8/ZJ/yz+59yrnh/wvY+H5JJ7b7V5kyeW/2h/Mo8L+F08JwXcEN09xHM8b/vE8v7lbdfh2YZhWzGtzVdj5fFYr24VakvEuLXZN/r0+5JVWivM1OMKKKKNSNQoooo1DUKKKKNQ1CiiijUNQoooo1DUKKKKNQ1CiirtvbpHH9quf9X/BH/fo1LOH+NGn/aPgf41DxzXPn6eI4LaBN7yyfaI+a+RPBv7RvxL+FvgbWfhlZaJZw2niVrrNrfaQ5uz9qt0tdkHQ/cjGzA+/X3ZJeTyT+fv8uT+Dy/4K+XPjNPNJ+2v8IJHmYyfadDHmb+f+P2sZn1/D2J5ZPCnmnwy+L3xZ/Y5i1e2j8JjQW8RvA7jxRosyO/2XzExHv2f893D1734P/bo/ap0e8up9J+GkNw80Sb/K8IXR+T+D7hrv/wDgtu7DWPhF8zf6jWP/AEZb198fAe7nudWu0muZpEGl2JCSO/H7usD7k/NjRv23v2oB4gvr2D4bQweIMTf6OPB16/mn+P8A5ab0evOvjl+1t+0J8QPCPi3SfGvgSDR9O1K1SDVLg+Grm1eBA8ePnk+5/q0r9bPBd1cyfGnVbd7mZ4w99iLe/lj95HXif7fkj3vwZ+NFss8kWzR7fZH53yP+/tKAPy4+EPx0+KXgf4K+J/CnhXw9Fqvg/UzfHUb+XRZLryPOtI4Z/wB+n+r2Qoj/AOx1rxrw74L8Q+L1n/sPQ9S1lYCiTf2faPOU3/cD7Bxnb+lff/7F8tzafsK/F628yaD974m3xiTy/wDmCwVT/wCCSN5Pbf8ACyRDNNEWv9AH7uTZ/wA/1AHzR8XP2mPiB+0toWlaFrGkadJb6NN9qj/sbTWSQfu44f3nL9o0r6Q+HfizV9P+HXhWyCQILXSYYNjwfOnyfx184fsmu8fibxGUd4/+JX/f/wCm8dfSNbwPjc6r88/YfynUaH4s1HUNYtLWbyfLmfy38tK7ivMPC/8AyMWm/wDXevT621Pja3xhRRRRqYahRRRRqGoU+3je4njghTzJ3fy0jqLzKv8Ah/UINP1ywupvuI/z0GkDK1j4X/8ACu9Gu9R0S9e41JIP38dwnnx+X/rP/adJ4f1hNU06x866h/tJ4I5J7eN/3iSf8tPkrv8AUI9R0u+1rZZTahBqifJJb/wV5V4s8D/8Kvj/AOEvmvUvNSup5I30yRPLjTz/APpp/wBM6+kyXOp5bPkn8B9HRrcnuHV0fdrJ8K6xJr+h2+otAlv53mfu438z7kmytbqK/a6GIhiaUKtP4ZHqli0vPI3o6eZA/wB+Oi7s/s+x0fzIH+5JUFFb8ut0IKKKK1AKKKKACiiigAooooAKKKKAO/8AgP8A8lMg/wCvG6/9p19A+KPDdl4w8M6loV+80dlqEHkTSWr+XJs3/wAD18/fAf8A5KZB/wBeN1/7Tr6U/gr+X/Eb/kc/9uR/9uP0LIP91/7ePm/Xr7xB8DPFFv4a8G2U174duXh1G6vNR0+S6dHeTZP++TZGiRxpH/uV71/wm/heTzvK8S6LJEo++NSh+5/33U3iZWk8J+IELbE/sy53/wDfl6/P7zIJNNgtbaBI7REj/g+//wBNJK/Ga+I+o7n3mFwzzFtLTk/E++X+IXhX7ieJ9E2f9hK2/wDi6Z/wn3hX/oZ9F/8ABrbf/F18BfZ4P+eCf9+6Ps8H/PBP+/ded/bUv5Dv/sVf8/Pw/wCCfW/7Q3x0tPht8D/Gfivwpr+g3viDSrOOeygkuYblGbzkT7iP8/yO9fj/APGv9pTxF+0f4utNb8c2unpLFp8emoNLt/JEMaSSSK6IX+/md+/zV9K/Ga3SL4P+MzsT/jwH8H/TxHX5/wDRi2OK/Q+HKyxeHlU5ep8RnmFeErRhz3NHVdKk0m6EbMskbrvhmT7kqc/OK/QLT/iZr0ek6aiPZeWljBH/AMev/TOP/ppX51r94V92af8A8g2w/wCvWD/0XHX7jwjp7b/t0+Gxv2D0jwX441fxB4igsrx7XyHSST93Bs/5Z/8AXSvRq8g+F/8AyOMH/XCf/wBF16/X6vhp88TxphRRRXYZhRRRQAUUUUAFFFFABRRRQBNaeTHveb95s+5H/fps87zvvf79R0VNtbgFFFFUBaknSe22S/69PuSVVoopJW2AKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFW7e3SOP7Tc/c/gj/v0W9ukcf2m5+5/BH/fqC4uHuH3vWP8AE0QzxH44fBzw98VPGFpq2uz6pFcw2MdokdlMkcezzJH/AI0f/npXn1n+yN4EuLyGF7vxCPMdI/8Aj9g7/wDbCvf/ABRbvJqsexHk/cVR0uzn/tK0/cP/AK+P/wBGV/EHF+bYzD55i6VOrop9j+gsly3AV8upVKkT4e+Avwz0T4leLda0/Wpb6K2s7N7mJrGZI33CaOMD50f+/X0E/wCyj4H8RbvNvte/tInh/tsP+kf7/wC4+/Xlf7INu8nxC8VBFb/kEycf9vUFfVH2Of8A54P/AN8VwZ7meLwmL5KMuiM8ly3CV8K51Inisn7I3gaOTY9z4g8z/r8g/wDjFH/DJXgT/n68Q/8AgZB/8Yr3yO3n1iPY6PHf/wAEn/Pf/wCzrNk0+6jk2PA/mf7lfPf23mP/AD9/A9z+ysv/AOfR8c/tC/CXQvhf/wAI6uiS6jL9v88TfbpkkHyeXtKbETs9er2f7KPgeeztZ2ufEAeeGCRx9sg48yPf/wA8K5n9syF4Y/BG9PL4vD/6Ir6O0+zn/srTf3D/APHjB/6Ijr3sXmuMhl+GqwnrPmv/AOBHkYXLcJUx+JpuOkOX/wBJPGf+GS/Af/P14g/8DIP/AIxT/wDhkvwJ/wA/XiH/AMDIP/jFe1fY5/8Ang//AHxR9jn/AOeD/wDfFfPf25mP/P78D3Fk+A/59Hxd8YvhVovgL4kaFomlSXzWOoW8NxLJdSpK/wA8zodm1E4+T9a9ruP2R/AlveTp9r8Q/u3eP/j8g7f9sK4H9pqOSP45eD0dcP8AYrbj/t6mr6r1Czn/ALSu/wBw/wDr5P8A0ZX0WY5pjKWFw1SlPWcdT5/L8twlTE4mFSPwSPC/+GS/Af8Az9eIP/AyD/4xT/8AhkrwH/z9eIf/AAMg/wDjFe1fY5/+eD/98UfY5/8Ang//AHxXzv8AbeY/8/vwPoP7HwH/AD6Pnnx1+zX4O8K+AfEGs2l1rUlzYW3nwpPcwum/egw+IR61y3wH+BXhv4meCbnVtYn1VLmO/e1jisZo402BEf8Ajjf++a+gPjFbTr8H/GweFx/xL+v/AG0jrgv2R4Xb4VXwRc/8TmT/ANEw19HQzPGSyqpiHP31I+fqZdhP7UhR5fc5S1pX7H/gfU9StbKO68QvJPMkf/H7B3/7YV4prPwDttH8P6vcnW/tGo6fDJNJBHH8mE8zP/oFfc3geT+z/F2k3UyP5CT/AD/+i6+O/E9j4g8O6143SHRrnWrXxH5nkT2v7zyf9Ynz/wDfyvW4ax2JxXtfrE+Y8XiTB4fC8nsIcp85UV3/AI0+FzeCPDem3t/qEP8Aat053WHdE9a4Cvtz4kK1tH1j7FvgnT7RZTcTQn8PnTnh6yaKANfWNH+x7J4JftFlNzDMB9fkfjh6yKKKAJMnPBr6G+BHwJ8N/EnwXc6xrE+qpdRag9rHFYzpGmwRo/8AHG/9+vntm+XBFfY37IsEkvwrvtiNJ/xOZOn/AFxhr57PsRUw2BlUoy5WfR5Dh6eKxihWjoT2f7JHgS4vIYXu/EOZXSP/AI/IO/8A2wrwz4D/AAx0j4leMNV0zWpL6K1tbRrlPsMyRvuE0cYHzo/9/wDSvuPS7Of+1bT9y/8Ar4//AEZXyN+yIkkvxK8TIi5f+ypuP+3qCvk8tzTGVsHiqlWesErH1uYZbhKeJw0Kcfj5j0T/AIZL8Cf8/XiH/wADIP8A4xTP+GS/Af8Az9eIP/AyD/4xXtv2Of8A54P/AN8UfY5/+eD/APfFfO/25mP/AD+/A9/+x8B/z6PDdQ/ZX8EQabe3CXWveZBazTIJL2HBeON3wf3HtXln7Pfwj0H4pQ+JW1uXUI/7P+zCH7DMkY+cyby+9H/uV9datZzf2DrP7h/+Qddf+iJK+eP2L43ktvG2xPMO+xP63FfQ4XNcZUy7E1XP3octv/AjwcTluEp4/DU1HSfN/wCknUf8Ml+BP+frxD/4GQf/ABimf8Ml+A/+frxB/wCBkH/xivbfsc//ADwf/vij7HP/AM8H/wC+K+d/tzMf+f34Hvf2Pl//AD6PF1/ZN8CL/wAvXiD/AMDIP/jFeJ/Gz4V6L4B8eaNpGkS3z2d7bQzs95MkrhnmdCPlRP7lfan2Of8A54P/AN8V8y/tMaTPd/GDwrGcRRrpttJNK/3ET7VN8/0r6HIs0xmKxUqdeeiTPBzvAYTD4VTpRO8b9jrwPLfTQJe+IRHA7iSR72H5EHf/AFFPvP2XvAjW7WFveeIlsVfeM3sP71/77/uK9x1zfJfXdrbQPHaJPJ/B9+T/AJ6Vk/Y5/wDng/8A3xXz8s6zG38X8D2Y5RgG/wCEeLL+yX4EUY+1eIf/AAMg/wDjFeOa18LdG0z9oiw8Dwy3o0a4urKNnknTzwk0MbviTZtGPM/uV9mfY5/+eD/98V8w+KY5I/22NHTb8/8AaGl8f9sIK+hyTNMbiZVvbT2hI8bNsuwmHhR9nH7cT0Oy/Zd8CLaLaXF34iexb5z/AKbD+6f++n7iq95+x74Hs5Nj3evSRuN8ciXsHz/+QK9j+xz/APPB/wDvij7HP/zwf/vivA/trMV/y9/A9v8AsfAP/l0ei0UUV/oqfzGFFFFABVu3t0jT7Vc/c/gj/v0W9ukcf2m5+5/BH/fqC4uHuH3vWP8AE0QBcXD3D73qOiitVpogCiiimAUUUUAFXY5E1JNj/u7r+CT+/VKiplG4BJG8cmx/v0VejkS+j2TP5d1/BJ/f/wB+qMkbxybHTy3Ss1JvRgFFFFbAFFFEcnlyRv8A3H30mB0lpodtZ+YlzbJd3aJvnkkuvIgh/wCme/8Av1Q1jR0t4fPhR7fZ87wb98ez++j1s3kH9pwTzW8M15Y3MyXaSQbHkhf+46PWbr0n2O1S1/1E7zvO8Ef/ACxR/wDlnXz1CpUdQ6THt7d7ydIYUeSR/wCCOuut7f8A4RHw79tmsvMvnfY+/wDgqh4CkT7VfImz7U8P7nzP8/7lQWPia9sLqe11ffcQP8kySfwVviJVKk3TXQIFv7PZ+MraR7ZEtNVh+/H/AH6ybTXtW0eOS2SZ49n7vy5E37Kk1fT08P30F1p11+4m+eGSN/uVm3l5NqF1JdXL753rajT9pvrDzM5iz3c13NJNcTPPO/8Ay0kqOiivUSSVkZBRRRTAKKKKACiiigAooooAKKKKACiiigAooqeCF53/AOeaJ9+ST+Ck3bcAt7d7h/8Anmiffkk/gqS4uE8vyYf3cH/odFxcJ5fkw/u4P/Q6qVkk3qxhRRRWwgooooAQ9Ks6X/yEbT/fjqselWdL/wCQjaf78dYVv4TGfPn/AAR3/wCS0eOf+xfT/wBLrepv2Yfk/wCCo/xD/wCwx4s/ncVF/wAEefk+M/jnd0/4R5P/AEugqX9mH/lKL8Q/+wv4s/ncV/JGI/j1/wDCfex+CJyX7WHwnuvjZ/wUp8SeA7C9t9JutcvLG3ju7hMxx/8AEtgfJRP92vXPB/xuj/4JhaVJ8JvEejf8Jzf6rN/wlK6lo92bVEjnRIBCUnh3hx9l3/8AbQVwfx4+JOj/AAf/AOCrWo+Mdf8AtA0bSL+xuLr7LDvk2f2XAPkT8axf2sJ4f21PiFpnjr4dRSpoVhpEOiTNrkkVnN9oheSRyI/Mf5Nk8fepnUhClD2ztCyNKNGdSclR+M1Pjl49g/4KHavpGoaHZf8ACFR+F4/sMq6pObnz3upJHTZ5MfRPLNX/APgjyhT42eOVPbw8n/pdb15n8K9Utf2Wo7228cCUvrE0F1bf2SI7r5YTIjh/3ibP9ZXpv/BHt/M+NXjpvXw4v/pdb15tOtVmsRy/wfd5P/bj0MVRp0IUbfxfth+zH/ylI+IX/YX8VfzuK5L9rz/lJX4j/wCv3Tv/AE229db+zH/ylE+If/YY8WfzuK5L9rz/AJSV+I/+v3Tv/Tbb19Rlf/I0o/8Abv8A6UeJiP8Ad5npvgfxxB4f02PS3spriSa6/wBZG6fJv8uOvV2r580v/kI2H/X1B/6Mr6Er+ncNI+MmBpBS1xHiDxB/wknn6Joj3VvqyT/6z/UR/J/rP3lc2YZhRy6lz1jnnOFOHPMPFHiBNYvrvwolrJHd3XlwJdyf6v8Av/79X/Cfh9/DelSWszw3G+fz/MjT/rn/APG6PDfh/wCx2No+pWsNxqyeZI93/rJP+/n+5W3X4bmGNqZlifbT/qJ8visV7eYUUUV5hwahRRRRqGoUUUUahqFFFFGoahRRRRqGoUUUUahqFFFFGoahRRV23t0jj+1XP+r/AII/79GpYW9ukcH2q5/1f8Ef9+oLi4e8k3vRcXD3km96ho1AxfG3iqDwL4N1zxHNbPfwaTD572qP5bv+8jj+/wD9tK5X4G/s8ar+214z0D41eHNQsfDWl+F9atdMl0nVWeaeZ7V4Lp5FdECfOkx/Gpv2gOfgT48H/UMj/wDSiCqv7D2uapof7D/xSn0vVdQ02aG58RXEclncvC6SR6NbyRyDZ/GCP0rGe59tw/Qh7Odf7Z9V/t9fsP8AiD9rjVfBs+i+IdN0SPQ4b5JBfQzP5jTvG642f7lfR3w4+GV94JvJprnUILvdZQWv7mF4/wDV/wDAzX5qf8E4fjH8QPHWj/EtvEfj3xR4gezl0uO1/tLWrqfyfMW88zZvk+T/AFcf/fFfoP8ABLWtQ1TxBqqXuoXl5GlsmyO4meTZ85rA+yOo0H4aXOlfEC+8QyX8MlvO8xS3WD94m/8A2681/aY/Zr1b43eCPHej6frNnpk/iSygtYpLq2d/J2SQSHfs/wCuH610vhHWr+T4xanZS395PZhrry7d53eNMPx8leKft0eMvEXhX4W/F2+0XxFrGiXmn6dayWtxpuoTQSQyGe0HyFH+Th3/AO/lAFP4K/sKeJ/h7+zz4x8Cah4p0q71XW/7UEd9DZzFP9KsI7VPM3/P8nl5qh+xn+wD4k/ZnfxQNU8UaVqn9rzadPH9ks5v3fkefvH7z18+vDP2TPi9488Tfsa/FHW9Y8deJdU1y1bXha6pda3cyXEPl6VBJH5cjyb08uQ+ZXxRZ/tXfF6+snt5/jF48tL2L5oZx4ovdkv+w/77j/foA9++Kn7Ler/8E+9Ps/EviTV9N8WW/iGR9KS10sSQSQ7MTb97p/0z2/jW94JH/Cb6fpVzD/on9oWv2pEf95s/d7/LrpP+CompajefB/4fRX17eXb/ANryE/a5nf8A5cYD393euc+APHhvwgP+oKn/AKBV0z5HPaMIU/bw+M7bS/A8+l6raXT3sMkcL+Z5ex662iiurU+FnP2gV09n4ftbOORLm1S8u0TzJ5Li68iCD/pn5n9+uajk8uSN0/5ZvvrsLy3/ALUgnntoJrywup47tJLfy5JIJP8AnnJG9Gp0UYGLrGjpbwedCj2+z949vv8AMj8v/npHJWTb2895PHBCjySP/wAs463vEEn2O0jtf+Ped55J3t4/+WEb/wDLOp/h/In2rUkTZ9reD9x5n+f+udGockJz5C/Z2f8Awhfhz7bNpyXGpO/lv5n8FV/s9l40tZHs0Sz1aH78f9+qOl+KL3T7qe11jfeQP+7nt5P4Kg1jT08P31pdabdfuJv3kEkb/co1Ojn9z+4R2fiDV9HjktUunj2fu/LkTzNlULy4n1CeSe5d7id/+WklF5eT6hdSXVy/mTvUNB586hxWueH30/xHP4rd4ZILV45/skf+sfZH5f8AuV1nhjxHH4p02S6hgkt445/I8uSpLi3S8gkgmRLiB/vxyJ+7euP1zw/qlvqsd1o7pp+kwpHJPb28/kb9n+s/d/7lfSZNnM8snaXwHqYLG8nuTPQR81GBWV4f8UWPij7ZJYedsh8v/Xx7Pv1qDLV+10MRDFwVajP3T6XcdRRRXYIKKKKACiiigAooooAKKKKAO++A/wDyU2D/AK8rr/2nX03HHvr5o+AcfmfE6D/ryuv/AGnX0tJJ/An3K/mLxF/5HP8A25H/ANuP0DIP92MrxlJ/xR/iBE+5/Ztz/wCiXr887P8A48YP9yOv0I8Xf8if4g/7Btz/AOiXr897P/jxg/3I6/Bc6+KB+m5JtUJqKKK+bPpzi/jZ/wAkd8a/9g8f+lEdfn4Pu1+gfxs/5I741/7B4/8ASiOvz8H3a/WeD/8AdJ/4z8y4q/3qH+EP4vxr7s0//kG2H/XrB/6Ljr4T/i/GvuzT/wDkG2H/AF6wf+i46/f+Ef8Al9/26fmuN+wdf8L/APkcYP8ArhP/AOi69fryD4X/API4wf8AXCf/ANF16/X6vhvgPGmFFFFdhmFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFW7e3SNPtVz9z+CP8Av0W9ukcf2m5+5/BH/fqC4uHuH3vWP8TRAFxcPcPveo6KK12WgHzV+0p8CdS+Knj6w1Oy1bTbGK10tLR47ozb9/nzv/BGf79eTf8ADG+t9/E2h/8Akz/8ar6+8Wf8hX/tglYVfxNxdnuMw2eYmjTkvi7H7zk2SYTEZfRnOJ8u/wDDGutf9DLof/kz/wDGqP8AhjbW/wDoZtE/O5/+NV9RUV8h/rLj/wCZfce5/q7gFtE+G9Q+Ed7YfFq38BNqFpJfTXMFul8m/wAj98iMP4N+PnHavWE/ZF1jXI0juPEuhx36fIk+26zMf7jny/8Ax+s7xNn/AIbO0j0/tLTP/RMFfTte/mec4vD06Dpy1nGL2PCy7KMLiJ1vaR+CR8vSfsY66kkiSeI9GR09rn/41R/wxvrf/QzaJ+dz/wDGq+tI5E1iPyJn8u7T7lxJ/H/0zkqhJG9u8iP+7dK8H/WXH7cy+49n+wcCteU+GNH+Et7rXxUvfAsN/ZpfWtxcwyXknmCFvJDl/wCDf/Ae1ehD9jfW+f8AiptE/wDJn/41Unw9z/w2P4h9P7Q1f/0XPX07Xt5znWLwVSEKbWsYvY8jKcowuLhOdSP2pHy7/wAMa63/ANDNof8A5M//ABql/wCGNNc/6GbRP/Jn/wCNV9Q1JHJ9nnjf+4/mV4f+suP7r7j1/wDV3A/ys+cfFP7Ldv8ACHQYbrW5NN8TXU03kXP+mz2UFkfL3/6zZ89Puv2N7rXfDOneI9J1fTdNt7y1juvsNxJPOiRv0dJkjIevaf2rLJPEHgXSpLWC51CxutaW9T7CiPJBOlvjy3R/vpXReF5P7P8AhL4MstiWc72PnvaR/wDLrG//ACzr6atm2Ip5fCvzLnPmcNldCpmE6HK+Q+XbX9i3xDfTxwW2v6PcTP0jRbn/AONV6poH7I+q/DPwab6e60G71x5PnaaOdxH/AMD8uvfvhnIn2rUkTZ9veD9x5n+f+udV9L8aajpd9Pa63vvIH/dz29x/BXhx4gxkqfNOSPankOE5+WETwHUv2PZPitYh7PWPD+j+JI/vx26T+XN/5Dr5ftPiH4u8CST6PFqEkYtXaFoJ0DhCh7BhxX6PahpaeG/Eek3ulXv+iTTxyQSRv9z95X5geLNavPEXiHUdRv5PNup5iZH219XkOZYjHSnHEL4D5jPcBSwMYTpfaKeueINQ8Sag97qV093dP1kkPNZ1FFfWHyIUUUUAFFFFADu4r174Yfs+6l8UPDcur2Ws6fp8UVy9qY7vzt5IRH42If79eRAHn2r7H/ZLbd8K70/9RiT/ANEw14ed4urgcE69LdH0GR4SljsUqNY4P/hjfW/+hm0T87n/AONUf8Ma61/0M2h/+TP/AMar6ior86/1mx/8y+4/Rf8AV3Afyny9/wAMb65gf8VNof8A5M//ABqvPdV+Et9pXxYtPAb39nJfXM9tEt6nmGH9+iOn8G//AJaDtX3HXzR4sU/8NnaEe39o6R/6Igr2sozrF42pWjVa92EpHh5plGFwkYezj9uJU/4Y51v/AKGbRPzuf/jdH/DG2t/9DNon53P/AMar6jj/ANXSV4v+suP/AJl9x7n+r2B35T5e/wCGN9b3Z/4SbQ8f9vOf/RVed6P8Ir7WPineeBodQs0v7W4uYXvJPMEJ8gOX/g3/AMB7V9y181/DG0kvP2yPEGzAiS+1h3f+BU8uevcyvOsXiqdedWS9yNzw8wyjC4edFQj8ciGz/Yq1+8Y7PE+iRog3PJ/pPyf+QqtX37Juty2v2C28T6Ktkkm/hboPM399/wB3X0/eXieX9ltv3don/j//AE0kqhXhf6yY+9+Zfcex/q7gXvE+Xf8AhjfW/wDoZtD/APJn/wCNVyXxO+AOpfDHw/b6teavYX0U1yLUR2nnbs7WbJ3oOPkr7Prxb9rnP/Cs9Ix/0F//AGi9eplefYzFY2GHqNWfkefmGS4TD4OdenE838N/spaz4i8O6bq8fiHS4Ib63S5SOYTB1D/SOr//AAxrref+Rm0P/wAmf/jVe+/C7/kl3hD/ALBsP866WuPEcQ4+jiKlJSVl5HXQyHA1aEKjifLx/Y31snP/AAk2h/j9pP8A7Srgfi18G774Qx6W17qdpqX2/wA0J9jL4XZszneg/vivt+vnD9sgH7P4PIGcG8/9o16eT51jMbjo0KzXJ6HDm+UYTCYKdajH3z7cooor+7z8BCrdvbpGn2q5+5/BH/fot7dI4/tNz9z+CP8Av1BcXD3D73rH+JogC4uHuH3vUdFFarTRAFFFFMAooooAKKKKACiiigAq19oS7hCSv5c6J8kn9/8A2Kq0VLVwCiiiqAKKKfa3aWl5BNNCk8aP9yT+Ok3ZXGGnx3Ul1HBab/Pf/nnJsrS0+8m8K6xOlzD5n8E0f99K6gSWWl6Pdatott5kk3/jlcx4fs4fEF9dQXc832uZPMSSvHVb2vPOa9w6OQn1jR/7P8vVtIf/AEH7/mR/8saTVtWsta0tJpk+z6qnyfu0+/VTTNYufD008GxJE3uk0D/6us6tKdCbd59NmZ84eXRRRXqmQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFTvdeZCkKp5afx/wC29QUUmrgFFFFMAooooAKKKKACiOTy5I3T/WJRRS06geK+JPBPir4Gw2Oq/s7aRceH/Et+0lpqdxpzm68y1+R40k+1PIiHen8Fe8/stz/s+al460O7spIJf2h7rTJZ/Ejyvqn2htReD/ia70f/AEbf5hn4Tj+5VDpVL4IfBLwd8HPjRP8AFTXr/UvDUF4l7I+reILyO00rz7mP/V+Y8cafP5kmz95X4vxdw7Rp0Z4zDPkbPo8txk3LkmfF/wDwUj5/bY+JOf79j/6breuA+Evij4kW1nJpHhC+ms9Labz5ZCiCFZNqJlndeD8ifpXu37U3hXSfjJ+2F4/8S6fqthq3hAzWXk6hpNylxDfOljb740kTjjnfj3rQ0/T7XS7WO1s4Ut4If3aRxp9yvzWFOFXDwhVhfY9X206NXnozOM1b4e6j47mtJ/HHiO58QS2qeXBFGiQxxj0+Svrb/gm74D0Lwx8VPGX9lWCW7voKIf8AWP8A8vSf36+SvGXxd0Lwa8lq7vqF8n/Lra8BP9+vaP8Agnb+0ho0Pxa8YT+J9Z0DwXpraGsdtNrWpxwCRxdR/u/MkeNH+Qv+VVOMaWGnThAUZzqVuecz3z4wQfCfwv4i1zWfg5HY6f8AG/8Ate4S+u9MSaS6y8j/ANpf6/8Ac5/1n/sleRz/AAd0zx9rKeNvGVreSfESeTzLvVp5n+0PJH+7j+RP3H+rRP4KXxcur+HPiT4s8a2ekaheeH73WL6ey1qSB5NNuYLqd/LeCfy/LdJPM+STzK0NH+MFlcSbNStX0/8A6aR/vI6+/wAkydUaaxS9+Z5OKxPN7hy+qfDfVPD99aTw/wDEwtEuoP3kf+sT95/zzr1bnNaGl3kF5JaTwzpJA7xyeZG/7v8A1lUPjJHqOjx/2v4VSG8jRJJLq3jg8/Z/00+SvvI5xDA05zxJ4tb9375xHiTxJdaxBAnhK9e4u0ffPHaJ9yP/AIH/ALdX/D/h+1s0tNQey8vVng/fyb/3nmP/AKz/AGKPD/hey0OSSe287zJk8t/tD+ZW3X5djcwrZlP21U+MxWKnXCiiivL1PPCiiijUYUUUUahqFFFFABRRRRqAUUUUg1CiiinqGoUUVdt7dI4PtVz/AKv+CP8Av0Fhb26Rwfarn/V/wR/36guLh7yTe9FxcPcSb3qGjUAoooo1IOA/aB/5IP49/wCwZH/6PgrH/Yv/AOTFvi3/AL3iX/0y29bH7QP/ACQfx7/2DI//AEfBXyx8HfjB8WvCPwT8V+F/CHhb+1vBGoNqB1LUv7EkufI86zjin/fpwmyFEf8A2Pv1zVD7/hz+BP8Axnvf/BLP/kB/Fj/rvo//AKBfV+l/wB/5GTWP+vVP/Q6/E39mH4xfFn4V2PilPhl4VHiSDUHtX1Fv7FfUTCY/O8n7n+r/ANZJX0Z4P/bI/a8028un0j4TzXE7Rqj/AGfwVdvsTf8A7FZn1h+lPgv/AJLhqn+/e/8AoyvCP+ChH/JIfjd/2B7X/wBH2lfJumfth/te2viy6vLT4TTPq7mYvbx+B7revd/9uuG+O37Tn7Snjrwf4u0vx38On0TSNTtY49Uu5fCt1aeTGHj2Pvk+59xOaAPQv2K/+TFfi3/108R/+maCvzor6J+DXxi+LXhH4J+K/C/hDwsdX8F6g2of2lqX9hyXPkedZxxT/v0+5shRH/2Pv187UAfph/wUuaK6+Dnw9WeXyZI9RfY+P+odb/u64P4aXk+n+BfC09rPJbz/ANkwp5kf+5XjX7SXxq+MHxO8I+HLD4j+Ev8AhH9Js7nzLG4/sKSw86TyEj++/wB/92iV7D8P/wDknvhX/sGQf+gVcNz5rPP4UD0Dw/4g1S81ywgmvXkjd/LeP5K9ArzDwv8A8jFpv/XevT66T4Ct8YVNp8d1JdRwWDzefN/zzfy6LO8SzvoJ5oEuI0f545P467vzLLS9Du9X0G1+0STf+QKYUYc5y+n3k/g/XJ47yDzN/wC7nj/1m+Op9Y0P+z/L1vR38yw/1nmR/wDLCoPDdnB4kvruC/nm+1zR+ZBJ/wBNKj0fWLrw3PdwbEkj3yRz28n+r8yjU0+wW9Y1iy8QaVHPMn2fWk/d/u0+/XO+XTqKNTnnP2gUUUUakBSSRpIkiP8A6t08uSlopCOH8QaHqmh+Wng+1mt45kk8/wCyfvN8n/LP79dfo/iTS9YfyLO9S4nRPMeOOrNcPqGlv4DjjuvD0F1cXcz+RP8AaE8/93/rP4K+pyXOa2Xz9nP4P6+E9jBY3k9yZ6JmlrJ8P65HqFjaJNdWv9pSQeZPbxv5ciSf8tP3daor9poV4YiHtKZ9JuLRRRXUAUUUUAFFFFABRRRQB6L8C7zzPiNaQInlx/Ybrf8A7cn7uvpAdTXzR8B/+Smwf9eN1/7Trp/2mv2kND+Dfwv8bXGj+L/C8HxA0mzjms9DvtQge78x3T5Psu/f9x99fzJx/QdTPIwh/JH/ANuPu8mqezwcpzL3xy/aW+GfwifUPC/jHxfb6B4gvtJkngsp7S6kDpMjoj744HT76PXwRafHb4cw2cKyeL7PeETP+iXX/wAYr5g/aA/aC8TftJ+NLXxP4uSxTUbWwj02P+zbbyI/IR3dPk3Hn9436V5Vkj72SK+Fr8N0MVCPt5yPbwueVsJz8kdz74/4X/8ADX/ocrL/AMBbr/4xR/w0B8Nf+hysv/AW6/8AjFfA+PrRx61w/wCqGB/nn/5L/wDInf8A604r+SP9fM+3fHHxL8KfEDwTrnhvwzrkWteINUhitbKyt7WfzJpPPQhPnTZ2r5F8W+A9e8DahDp+vabNpl5LD9ojinGN8eWG8fip/Kq/hLxZf+C/E1hrlgYjeWMwmi85N6Fh6ivf9NHhj46eEdR8SeNtbtrTxTp8E9lY2Fld29qJkjg8yDELIXkeSZ3TINephsJ/YXu0veo/+Tc3+RyV8Qs7d5+7V/8AJT5kX71fden/APINsP8Ar1g/9Fx18Wa74V1fw3MiappN7pbSfcF7bvCW9/nAr7T0/wD5Bth/16wf+i46/bOEJRl7acP7v/tx8NmEPZq1Q6/4X/8AI4wf9cJ//RdevZ5ryH4X/wDI4wf9cJ//AEXXvnhfwu+uXHnzfu7FP/H6/UKdSFOEpzPD1M3S9HvdYk2WcHmf35JP9WldXp/w7hj+e8unuH/55p8iV1ltbpZwpDCiRon/ACzjrK1jxhZaPJ5H/HxP/wA8468ueMr4iXJTOjkhAfb+E9It/uafDJ/v/PU8nhnTJP8AmH2v/flK4y48f6jJ/qba2t4/+mnz0yP4gapH99LWT/gFL6pinr+ovaQOkvPAemXH+pR7ST/pm9cnrHg+90iOSZP9Mg/vx/6xK6HT/iJbSSbLuB7f/ponzx11NvcJcQI8LpIj/wDLSOoVbE4T4w5IVDxyiuv8YeF0t0k1CzT93/y3jj/9GVyFfQ0K8MRDngc84ezCiiiuozCiiigAooooAKKKKACiiigAooooAKKKKACiiigAq3b26Rp9qufufwR/36Le3SOP7Tc/c/gj/v1BcXD3D73rH+JogC4uHuH3vUdFFarTRAFFFFMZ86ftHSfFePx1p/8AwgaagNK/stPtH2VEeLzvPn/v/wCxsrykTftI55XWf+/EP+FfV3iz/kKf9sI6xa/iLi7Nlhs7xNP2EH73WJ+9ZNlftcvoz9tP/wACPlHxV48+OfgXTIbzxBf6lpltNJ5SSzwxfMw7dKv6XrX7QmuaTa6jYS6tcWV5D50FwkMP7yP16V2H7X//ACTnQ/8AsKP/AOiRXpXwsj/4tb4P/wCwTDXj1MdTp5dTxn1WHPOX8ppTwVSpmE8J7afJCP8AMfM9x8PPjTeeL4vFUukao+vJJHOt7sXf5ifIh+v7v9K6QTftH+msf9+If8K+maK86fENSpb2lCDt/dPQhkMIfw60/wDwI+SvF3xJ+NvgMWn9v6jqGmfaxJ5XnQQ/vNhw/wDBXXw6n+0JqenJ9oOumcx77eeOGHY6bPuPx6VV/bJ/1Pg3nHzXn/tCvo3T4/8AiV6b/wBeMH/oiOvQxGYU6GCoYhYaF5832f5TzqGXzq4ytQnWn7nL9o+RdP8Ah78bNM8WzeJ7TSNUg12aSaR70Im9pH/1h+vz/rXUed+0h6az/wB+If8ACvpWivOnxDOrrUoQf/bp6MMhhT/h1p/+BHyR4k+KHxr8I61aaVquqX2n6jdRo8MEsEPmOjNsT+DuU/Sujkb9pGNmR11nKdR5MP8AhUX7S3/JavB/PP2O2/8ASqavqz7ZBp/iOSeaBLiNJ5N9vJ/HXq4zHUsNQoVFQh+8/unnYXBTxFatCdafuf3j5J1Xwp+0F46e1068sddvVWbfBHDshxJ9z+DH0rrPBL/tC/CrU4NMuNN1VtKtYfsr2caW0myD0R6+yPM07R/D93rfh6yS4km/8gVyfhfT7XxRqV/Bfzzfb5k8yC4/6aVhUzmcYwo+whb/AAlQyiE5Tq+2n/4EfO3xGt/jr4PsX8X6HfatH4fhT7VNK6W2bL/7Cs+P4sfFD4veAf7a0WbWpdcs5vsrSWsEHkTufv8A8HXy5K9m8feJLrwX8M/HUZhSeNNOeOe0k/1e/wAyOOvKf2T5PM+GOoOFVQ+sT/u1/wCuMNdSx9F5dPF+whpL+U5/qE/7QhhfbT+H+Y4+1uP2kfOj8qPWvM3fJ+5h61823XmC4l87/W7vnr9MdL/5Clp/13j/APRlfmnqBI1C5z13t/OvU4dx/wBclVXs4Qty/D+p5fEWC+qRp/vJz/xFCiiivsz4sKKKKACgUUUATv8ANjbnHfNexfCWT4tx+F7j/hA0vjon2x/O+zxxPH5+xM/f/wBjZXjbMWHBJFfYv7JLbvhXfEf9BiT/ANEw14eeYn6rgpVeRTt0lsfQ5LhvrWMVPncP8JyMbftIvIqIusGR+g8mH/CuY8K/FT4zeNtQmsdC1W/1K9t4GmmggghyibwhONnTLp+dfY2l/wDIUtP+u8f/AKMr5N/ZJYN8TPEvP/MJl/8ASmCvksBjaWKwtevUw0E4f3T63HYKph8VRoQrT9/+8aRn/aS7LrP/AH4h/wAK5y8+HnxsvvF0Pie40jUpNejkjkS+2RbxInyIf/HMfhX1xRXkw4gnTv7OhBf9unozyGFT+JWn/wCBHzRcXX7RdvDPPMdYjggjZ3kMMPyInU9KwfB/xI+Nvj2G+l0DUtQ1MWZjE3kwQkpv37P4P9l6+q9Y/wCQBrX/AGDrr/0RJXz9+xec2njXn/lpZf8AtevSw+YU62Br4t4aF4cv2Tzq2CqU8ZRoe2n7/N9oq+d+0h6az/34h/wrHi8H/GbT/EX9u6boOqWWrTO8812ER5Jpnz5jvx/tv+dfVtFeZHiCcNIUIa/3T0Z5DCfx1p/+BHzUJv2kv7usf9+If8K5zxL8VPjR4N1S007WtWvtPvbqNZobeSCHzGR22D+Dvs/Svrivl79qP/krXhbnn7Db8f8Ab1NXq5RjqeYYh0amGgrL+U8/MsDUwOHVaFef/gRdkb9pFJHRxrIkTt5MP+FYnizwj8dvHWnw2Gv6dqmqWsE3nJHIkQ2P9zt+VfXWqf8AISu/+u8n/oyqleTT4gnSlz08PBW/unpyyKNVck68/wDwI+YdKs/2hdF0m00zT7bWLays4fJhgjhiPlx+nSsW4+KPxpsfGMXhafVr9dekeNEsPIh8ze670H3Ovz5/GvrivmbxR/yepo3P/MQ0v/0RBXsZbmNPMJ1PaUIaQlP4TzMwy+eBhD2dafxxj8Qom/aO7jWP+/EP+Fc/4z8B/Gvx59jHiHSNS1YWm/yfkiGzf9/7n+5+lfXFFeVDiGdCfPToQX/bp6MsihXhyVK0/wDwI9K8xI/vvVqD7LDb/arl02fwR7/v16FoujH9k83WtavdHxKmtmOygg02PyJEdPMn3vvNav8Aw2NpUn/Mq6z/AOBsP+Nfvz8WKcVavhuSf+L/AO1PzFcI1Kn8CfPD/D/9seQ3GoJO+95kpn2iH++leu/8NjaV/wBCrrP/AIGQ/wCNH/DY2lf9CrrP/gZD/jS/4i9hV/zDf+Tf/aj/ANTcV2/r/wACPJY/9I/1P7z/AK51G7JD99/Lr51/4KbfF60+Kl38NJLTTLvTTZW2oFvtkscm/fJH/c/6519s2elf8Mo6hfatqkx8TQ628enxwabH9leF4RJP5j7zXfU8T1ToU8RHDe5P+9/9qcNHhuc6k6E5+/H+7/8AbHmP2iH++lJ9oST+NK9d/wCGxtL/AOhV1j/wMh/xrivjZ+11pV18IfF0R8Mauimw5ZryHA/foO3+FcS8WqNV+zp4bX/F/wDanZ/qjiVrOX9f+BHNmQmoxPD/AM9Eqj+zboo+LXgZPitYXH9naXoepzTvpdwnnTzfY/Lnfy5Ewnz5r2pf2xNKKhv+EW1k7/8Ap8h/xpf8RYlh3yYrB8n/AG9/9qH+qrru2Fq8/wD27/8AbHkn2iD++lH2iH++leu/8NjaV/0Kus/+BkP+NH/DY2lf9CrrP/gZD/jR/wARfwv/AEDf+Tf/AGpX+puK7f1/4EeQxyRySbEdJKkkj8v7/wC7qX9qr9pfTPG/7NvxF0WHw9qVpJc6bGiT3F1C8afv4PSvL/2B/hHda98CrfxrFq1vbwaN4iur6Sx8h/Mm+zR2k/lo+/8Aj8v0ruXigqmF+sUMNzf9vf8A2pxPhv2db2NefJ/27/8AbHpH2iD++lH2iH++leuD9srSriMP/wAIrrPz/P8A8fkH+NL/AMNjaV/0Kus/+BkP+NcP/EXsN/0Df+Tf/anZ/qdiu39f+BHknnJ6Uz7RGn8aVwfw48aWX7Sn7SHjXwdpcFzoV5DcapqT3186zp+7nxs2J/v+tfS2h/FmL4IxW3w5l0XUPEWpaQmyS9050ggmM3775Ec7/wDlvWj8Uq1P+PgOX/uJ/wDak/6sQml7DEc8/wDDt/5MeS+fB/z0Sun0eW20/wAG32oJGlxM/wAj/wCxXpOqftP/ANjw+feeCNcjj/56fbLWT+tVdP8A2w9NkvoIIfCusSec/l+X9pg/xrKt4qUaloSw1v8At7/7UmHCeK+OEv6/8CPKNE8SPol1vhdJEf78G/79dAmtaLpvmahY6fNHdSf3/uV6F4h/ao0nw3qr2r+G9ZuPk8z5L2Cvz8+AfhR/jZ+3l8WBZXCaSby617Uk+0w+ftzdk7CF/wCun6VsvEWhjo1Jww3vQ/vf/amFTh2ph5w9vPkhP+v5j6SkvI5JpHmdPMd970v2iH++lepaX8abT4B6fB8P73StQ1i70NNkmo2U0cEMxm/ffIjnf/y2q1/w2Npf/Qrax/4GQ/41l/xFuhTfJPDf+Tf/AGp1LhHFVFenLT+v7x5FHIjf6t6JJEX/AFj1xn7YH7XWhm98H3E3hvVB/o96AJLyHu6egPpX0Dp+gSfslXl1rWq3J8TQ6x5elR2+mx/ZXhdPMn8x95rR+KVRJVHgv3b+17T/AO1H/qrD4Pbfvv5OX/7Y8y+0Qf30o+0Q/wB9K9d/4bG0r/oVdZ/8DIf8aP8AhsbSv+hV1n/wMh/xrP8A4i/hf+gb/wAm/wDtR/6m4rt/X/gR5F9og/vpT4/3/wDqf3n/AFzr1r/hsbSu3hXWf/AyH/Gvhj/gpv8AFyy+K9x8NpLXTLvTDYwahv8AtkyOX8ySP+5/1zrtwXipRxlf2FPDf+Tf/anHieFsThaPtqn9f+TH0TJIkX338ugzQH+NK9QtNDk/ZRvr3XtSuT4lh1iSPS0t9OjFq8Lpmbf85q9/w2NpX/Qq6z/4GQ/41xPxapU/dr4bkl/i/wDtTqjwlUq60Jc8P8P/ANseQ/aI5P40p/movQV0nxu/a60q9+Dvi6M+F9YUCw5JvIcczoO319K4r9m3w7/wubwPD8T9Nu/7K0/RtWaSTTLhPPnm+xiOd/LdMJ8+fStP+IpTmva0MHzw7+0/+1D/AFVhT9zFVuSf+H/7Yv8A2iD++lH2iH++leuj9svSpxv/AOEV1n5/n/4/IP8AGj/hsbSv+hV1n/wMh/xrP/iL+G/6Bv8Ayb/7Uf8Aqbiu39f+BHkLXEP99KFkSRtiOklevf8ADY2lf9CrrP8A4GQ/415V+1d+0tpvjr9m34i6JF4f1KzkutMjRJ7i5heNP9Lg9DW9Dxaw1apCmsNq/wC9/wDamFThHE04e0a/r/wIif8Adff/AHdRieEfxpXm/wCwN8JbnXvgba+NodWt7e30bxHc3r2P2VzJN9mS0m8tJN/8fl+lfUy/tlaTMob/AIRXWfn+f/j9g/xp4jxUWDqThWw3/k3/ANqKjwvUxUISoz5/+3f/ALY8i8+D++lP8xK9Yb9sTSmGP+EV1kf9vkP+NfL/AID8fWX7RH7SnirwXplrc6FqEl7qd79uvnWdE8mR32bF/wAaxXitPEJvC4Pnt/e/+1N/9VXTf+1VeT/t3/7Y9E+0JH/GlL9og/vpXqWl/Gm0+AenwfD++0rUNYutDTZJqFlLHBDMZv33yI53/wDLarX/AA2Npf8A0Kusf+BkP+NT/wARcor+Jhv/ACb/AO1BcI4qSvTlp/X948j+0Qf30o+0Q/30r13/AIbG0r/oVdZ/8DIf8aP+GxtK/wChV1n/AMDIP8af/EXsLt9W/wDJv/tR/wCpuK7f1/4EeS+W/l7wnyVH9oROrpXzN8EvC8vxl/b6+I6WkyaT/aGo+IL+P7SnnbMzSPs+X/f/AEr7s0r4zWf7P9nF4AvtL1DWrrRU2SahYzRwQTGYed8iOd//AC2rtxfif9TmlUw3ufzc3/2px4bhueKjanP3/wCXl/8Atjyw3EA/jSiKRZPuPXr3/DY2ldvCus/+BkFfN37X37XGh/bPCFzN4a1X/j3vAnmXkXd0HbPpXJDxYhiHyYXCc8/8X/2p2f6o1Kfv4qXJD/D/APbHZSSJGPnb8a2P28rP+2f2KfDtjuVHvdQ0GBJH/wBuN67XTdDl/ZPkn13U7n/hJIdbMenRwadELV4dmZt77zXV+Df2n9L8QajJjwzqqeSnmfPcQv8A1r5HO+OIZ17GFej7GcP73N/7addHh6eGhOdCXPA/OjRYdM0HS7WwtJIUtbZNiIZK81+M/wAVRosP9jaNcqb2ZP388T/6lPT/AH6/Y24/aLsbeCSd9Dv/AC0TzH/ew/4150n7Y+myf8ylqsk7/wDTa1/xrw63EGFp7jp5Hiqmtj8SfCvhnVPGWqGy02AzzH53d/uIP7zn0r2zw/8AALw9pcPma1qL384+/GknkR//ABdfcP7anjSHxhJ4MnisprOS1stS3+e6fPvkg/uV9Ea54iX9m8/2nfRnxCdSkj05ILGPyPJdEM/z7zXZWzSNOjCvy+5Mwo4Gc606H2zC8UeF9IvP2KvAtlIkn2BNH0BE/fv/AARx+XXy14g+FaRxyT6Pe/aP+nS4f95/2zr6qj/bSsf+hP1L/wACIP8AGn+L/wBqbRPEnwz8TvBoepI8Fl+8t3lhEn30969HJOM6OXUfY0Xz+/8A19krGZBiZ+/WR8Y+D/FF14X1iODf+4eeOOe0k/66eX/33X1Z4bj0+Oxke2vftkF1/wA9KwvAXhRPiroUfxP04to9voeqO91ZToJHuktSk/7t0Oz5/uV6X4P/AGstG1zUHgTwxq0BmTz08y5g/wAa5sw4sniMT/tsOT+T3v8A7U8ipwpPER5sLW54f4f/ALY8P8SWcGh6xPa+enkP+8g+f/lnWb9rg/57p/33X0r4w/aQ0zR7FL1/Dep3Gx/L/d3UFcv/AMNgaP8A9ClrH/gTB/jXnT4mw1P3P6/9JPMhwBiq/vwn/wCS/wD2x4tbyJcSeXC6SSf8846fJbvb/wCuR4/+ulXP2rP2l9N8cfs2/EbRbbw3qdnJc6ZGnnz3ULxp/pcHpXln/BPn4U3PiX4JzeNIdVhtYdF8T3N1LYNAzvOlrBaT7I3DiutZ1CphPrFGHP8A9vHM+B5063sa9bk/7d/+2PQPtcH/AD3T/vuj7XB/z3T/AL7r2iP9sbR7gB/+ES1n5/n/AOPm1/xqX/hsDR/+hS1j/wACYP8AGuD/AFpw39f/ALJ2f8Q6xv8AP/5L/wDbHi3+s+5UclxBHJsd0j/4HXHeC/Hln+0T+014t8Gada3Gh6g17qd6b6+dZ1TyX+55afT1r6T0/wCMlj8BbGH4f3+kX2vXehpsk1CyeOGCXz/3/wC7Rzv/AOW+yrlxDUpP9/R5If4v/tSf9QfaWVHEc8/8P/2x479rg/57p/33R9rg/wCe6f8Afde2/wDDYGj/APQo6z/4Ewf40f8ADYGjf9CjrP8A4Ewf41j/AK1YX+v/ANkr/iHWN/m/8l/+2PEvtcH/AD3T/vurf2efyPP8iTy/+emyvYv+GwNHzx4R1nH/AF8wf418E/Bjw8/xm/4KBfEeOznXSWvr7Xr5PtSedt/eSPs+X6/pXbh8/hjIVHRj8Bx1+A6+HnD29bkhP+7/APbH0TJcQR/fdI/+B0fa4P8Anun/AH3XsGn/ABlsfgRp8HgC+0m+1260NNkmoWbxwwy+d+++RHO//lvsq5/w1/pH/Qpax/4EQ/41xPimitJx/r/wE64+HeKkr06un+H/AO2PF45EuPuP5n/XOiSRLf77+X/10rmf2wf2vND3eD5pvDGqKoivR5f2yHu8P/TM+lfQWj6Cf2UWu9d1i5PiaDWEj06C002DyHR0zPvffWv+sNRJVPY/u39vm/8AtQ/1BXwfWf338nL/APbHlVvJa28H2q5nTy/4I9/36guNUS4k3vOn/fde0Sfth6PJ/wAylrP/AIEwf40v/DYGjf8AQo6z/wCBMH+NY/61YXv/AF/4CP8A4h1jf5v/ACX/AO2PEvtcH/PdP++6nt43vPM+zJ9o2f8APP8AeV7Mf2wNH7eEtY/8CYP8a+Gv+Cnfxfs/ivD8MTaaVeaZ/Z66pvF3JG2/zJIP7n/XP9a7cFxFRxlaFCmcuJ4DxOFo+2qT/wDJf/tj3ySRLf8A137v/rpUf2uD/nun/fdet2GiH9lOafXdUnbxLDrawaclvpsHkPE6Zm3/ADmtL/hsDR/+hR1n/wACYP8AGuN8UU6btWhyT/xf/anTHw+r1daFXnh/h/8Atj568baDa+PvBuueGZtQ+yx6tD5LzQIjun7yOT7n/bOvBvEXxfv/ANjvwL4h+Dmh2dv4o0rxJZXd9Jqt6Z7W4t2vbQWroscb7PkWENk19l/Gb9rfRrn4R+Ml/wCEX1WAf2a+c3MP99B2+tcD+zHpP/C7/Bc3xM0qb+yrDQdYcSafcx+dPP8AZUgupPLdMJ8/mbOlOWfVK8Pa0aPPD/F/9qezl/Ck8tm4YrE8n9zl/wDtj4b/AGYP2s9W/ZlsfFFrpugWWtjXZLV5PtlzPCE8jz8YMcif89z+Ve/eG/8Agrr4v8N3FzNB8PtDczIiH/iYXvb6yV9sR/tnaXPGH/4RPVfn+f8A1tr/AI0v/DY2l/8AQpar/wB/bX/Gs/8AWbCnuf6vY3sfE9j/AMFdPGGm69PrEfw90Pz5t+9f7Qver4/6af7ArjPjZ/wUi8SfHDwT4u8Pah4O0nS08RWsdrcXFreXTvGEkjcEI77P+WI7d6/Qs/tjaX/0Keq/9/bX/GvMf2mv2otN8Y/s8/EXR4vDmpWb3ujPAk88sPlp+8j/ALhrWlxFha1RU0Z1MhxVOHO0fn/8If2vtX+D/wAEfFXw0sfDun6nYa+18z6hcXM8ckP2q1S1kwiOEPyJ+tfO/XA6V+k//BPj4W3mvfA+bxpDqtvb2ug+J7i6ksmtfMknS2gtJ3RH7b+B+FfV0X7Z2kXEe/8A4RPVfn+f/W2v+NaYjPKWErTp1YBRyipioRlQlzn55eMf2grv9tTS7Dwr4jtLHwjY6CDqUVxYzzXElw+yC32fvpP7grrtBsINB0HTtMS6+0JZ2qWqT/6vfsr7ck/bG0iT/mU9V/8AAm2/xr5n8D+PrT9on9pzxh4L0+1uNCvHvNTvjfXzrOieS/3PLT6etZU+IPaJvC0ee397/wC1PLxnCtSvO2NxHs4f4eb/ANuOb8L3lrH4m01Hnj8zz/79eo/a4P8Anun/AH3Xsuh/FyH4Jx2vw5n0XUPEWpaOmyS9050hSYzfvvkRzv8A+W9bGoftNJo8Hn3ngXXI4P8Anp9stX/rVw4mo/bh/X/gJ4UuAq9Sd6dX/wAl/wDtjy7R7ix0/wAD3+ookNxPN+7f/YrC8P8Aij/hH7rfDOkkD/ft9/369h079r/SpL1IIvCWs5mfy/L+0wf41b8QftSaN4b1R7J/DGrSfJ5n7u5g/wAaP9ZsNy839f8ApJr/AKg43m5ef/yX/wC2PO7fWNE0uOTUbDS7r7W6fJ5ifu64y4vEknkeadPPd/MevnP4M+HZPjN/wUB+I8dpOukG+v8AXr+P7Unn7P3kj7Pk+v6V91aX8XrT9n20h+H+paTfa/c6Om+TULJ0hgm879/9xzv/AI9ldWLzqGDmvaQ9x/a5jno8FzxsbU63v/ycv/2x479rg/57p/33UkciXH3H8z/rnXtJ/bA0ft4R1n/wJg/xr5u/bA/a80RW8HzTeGNUxtvR5cl5EP44P9g+lc1PiWGI/d4WHPP/ABf/AGpf/EPa9N8+Kq8kP8P/ANsdRJIlv99/L/66VH9rg/57p/33XrGm+F5f2U5Z/EOqXR8Sw6qkelR2emweQ8L8zb97n/YrV/4bA0f/AKFHWf8AwJg/xrH/AFohT/jw5J/4v/tR/wDEPq9T+BW54f4f/tjxL7XB/wA90/77o+1wf890/wC+69t/4bA0b/oUdZ/8CYP8aP8AhsDR/wDoUdZ/8CYP8aP9asL/AF/+yH/EOsb/ADf+S/8A2x4zbxveeZ9mT7Rs/wCef7yo5JPs/wB9/L/66V4J/wAFOvjFafFSP4Ym10q7006euqbxeSo+/e8H9z/rn+tfaNh4e/4ZYu5/EWqXTeJYdUSPS47TTYPIeJ+Zt/zn/Yr0Kmewp4eniIQ9yf8AeOWjwLOVadCpW9+P93/7Y+b9Yt7XwXdT+KIb1Lyd55I/s9x+7j/f/wDTSuv8I+JD4k8Pwag3kxyT+Z+7t5N8fySbK90/4bD0f/oUtY/8CYP8a434yftc6Pd/CXxjF/wi+sQx/wBmuMm8hxy6Dt9a9XJ/EKGXz9nCHP8A3eb7X/gJ7uG4LxuH1qT9z/D/APbHLeYfWmedD/z0Ssn9mPQP+F8+DJfiNpVydGsNE1x45NPuY/Pnn+ypBdSeW6YT5/M2dK9yj/bI0q4USjwtrJE3z/8AH5D/AI19W/FaWH/3vB8n/b3/ANqdn+qrrv8A2Wtz/wDbv/2x5J9og/vpR9oh/vpXrv8Aw2NpX/Qq6z/4GQ/40f8ADY2lf9CrrP8A4GQ/41n/AMRfwv8A0Df+Tf8A2pX+puK7f1/4EeQxyRySbEdJKkkjeP7/AO7rQ/aa/ae0zxr+zr8RtHi8N6taSXukvGlxcXMLxp+8j9K8c/4J9fB+68ZfBSbxbb6va2UOkeJ7maS0e0Z3n8iC0m4cOMV3rxQ9phPrNDDc/wD29/8AanH/AKtOnX9hip8n/bv/ANseofaIP76UfaIP76V64v7ZWj3Sib/hFdY/ffP/AMfkH+NL/wANjaV/0Kus/wDgZD/jXB/xF/Df9A3/AJN/9qdf+puM7f1/4EeC+NPjtd/s56E/jqy0e28QXEM6ab9juppIU8ufzP3m9P8ArnX58ftDfGK4/aG+MGveOrvTIdEudWFtvsYZnkRPJt44Bhn5PEf619z+BviBZ/tFftN+LfBemWtzoWoNe6nem/vnSZF8l/ueWn09a+k9N+MVv+z7aQ+ANQ0261660lN8mo2XkwwTed++/wBW53/x7K+JzjiyWOxn1zFYb2Xu/wA3N/7ad9DIrUlRw1Xnn25T8MduOM0Yr92P+GxtL/6FLVf+/tr/AI0f8NjaX/0KWq/9/bX/ABrx/wDWbCm3+r2N/lPwqdSMDOaXazcDk+wr91f+GxtK/wChS1XH/XW1/wAa/Pv4P6LJ8X/+ChXjqOzmXSjqmq+IbtGuYfP2Z899mF+td+Gzmli4zdFXlA5q2U18LOHt/cjM+LGjZeorQ0bUH0fVLS8Vd4tpkn2Ho21s1+3em/GS3/Z/sYPh/qGlXWu3Wkpvk1KyEEME3nfvvuOd/wDHsq3/AMNjaX/0Keq/9/bX/GvOlxHhn7k0dccixbtUoo/Ni0z+2NNd3eo3P/CKf2BJ5aR2sT33nfankf8AjkTYE8mu48yCzgggedP3KJB5kn7vfs/d16/+2F+2Fo8f/CG3EnhbVD+6vR5f2qGLq8H9xPavdtD0ST9ltR4o1SY+IbTXFh02O0sYPIeF/LN15kjuf9jZXq8N8VT4ddapCjz4afwe98PL8X2eaXvHRmWVQzGNOnOf+0/b/vfy/wB34T5Z+EckGqePrC1huofMmSeP7/8A0zr7BsoLXT7VLaF1jRE8utfwn+1JpniDUpNvhjU7fyE8zzJLmCuruf2h7G3t5J30LUtiJ5j/AOkwf419i/EijiI39j/5N/8Aanzv+rOJpz5Lf1/4EeK+NPFiWP8AxL7OZI53++/9yuQ0nTZtam8mxT7Q/wDHJ/cr1OP9sPTpP+ZV1iR3f/n8g/xrxr9tzxpa+NLXwY66dNbyWVrqW/z3R9+/yP7n/XOu/LfEijWk6NDDa/4v/tTDGcOVsLDnqS/r/wACPRdP+H9lbx/6fO9w/wDzzj+RKtyeC9B8v/U+X/22eut1LWF/ZvtYtbu1k8QLqSwaclvYp5HlP5Zm8ze5/wBis/8A4bXsP4/CWpf+BkH+NedU8S/Zv9/Dkn/i/wDtTqp8L16nv0FzwOI1TwJ9nSR9Om+0f9MJPv1h6Hrk/h+6k2f6v/lvBJXrHiz9qTQvEnw18TyxaFqaSR2Um+3eeHf/ADrzvwD4JT4zaAPiLpUp0SPQtU/0q1uk8yS5+ypHP99P++K7l4kVqlDnjhPaQ/xf/amEOHYwnyV6vJP/AA//AGx3NnqFrqlik8LpJBMleW+INPTRNUntd6CD/WQf9c69m8G/td6Tr2oPAnhzVrf7TH56eZcwVr+MP2lLDR7FL1/D+rSRo/l/u7mCuTDeJdGj78KP/k3/ANqaT4XxXPyT/r/yY+dftEH99KRJEkk2I6SV69/w2NpX/Qq6z/4GQ/415l+0x+09pnjb9nX4i6NH4f1Wze70lo0kuLmGSNP3kfpzXdQ8W8NWqKmsN/5N/wDaiqcI4mnD2jX9f+BFJ4ni+/8Au6j8+HP30ryv/gnn8Irvxh8E7vxfBqttaQaL4rmle1a3Z3m8iC0mGxxX1in7ZWjXSiX/AIRXWT53z/8AH7B/jV4jxTWDrTp1sN/5N/8AamdHhipiqUJUZ8//AG7/APbHkXnQ/wDPRKf5i16y37YmlMMf8IrrI/7fIf8AGvl/wh8SrL9oL9p/xJ4H061udC1C61DVLr7dfOk6J5G99nlp/wBc/WoXitPEJvC4Pnt/e/8AtTf/AFVdN/7VV5P+3f8A7Y9F+0JH/GlH2iD++lep6Z8YbT9n+wg+H99pd7rt3pKb5L2ymjghm879/wDIjnf/AB7Ks/8ADY2l/wDQq6x/4GQ/41l/xFyiv4mG/wDJv/tQXCOKlrTl7n9f3jyP7RB/fSj7RD/fSvXf+GxtK/6FXWf/AAMh/wAaX/hsPSv+hV1j/wADYP8AGn/xF7C7fVv/ACb/AO1H/qbirbf1/wCBHknlv5e8J8lR/aETq6V8v/CTQP8AhcP/AAUK8dR2k0elHUtV8Q3kf2lPO8v/AI+H/g+tfeml/F6z/Z9s4fh/f6XqGv3elJvkvbGZIIJvO/f/ALtHO/8Aj2V24vxO+pzSqYb3Gvi5v/tTjw3Dc8VG1Ofv/wAvL/8AbHlZuIB/GlEciyfcdK9e/wCGxtK7eFdZ/wDAyCvnP9r79rnQxD4Pnl8M6oo/4mARJbyL/ph6Zrlp+LEMQ+TC4Tnn/i/+1Oz/AFRqU3z4qXJD/D/9sdf5iR/ferVv9lht/tVy6eX/AAR7/v133h3w9/wy3E3ivVLo+IrXWoYNOgsdNg8h0fBm8x3eT/YrV/4bG0qT/mVdZ/8AA2H/ABrN+LEIfu6+G5J/4v8A7UFwjUqfwJ88P8P/ANseS3GoJO+95kpn2iH++leu/wDDY2lf9CrrP/gZD/jR/wANjaV/0Kus/wDgZD/jR/xF7Cr/AJhv/Jv/ALUf+puK7f1/4EeSx/6R/qf3n/XOo3aOH77+XXz/AP8ABTz4zWXxYsfhebTS7vTTp8mqb/tkscm/f9k/uf8AXOvsTQNBb9llI/Euo3Z8RW2sWsGlx2mmwfZXhfyzN5nmPJ/sV3VPE9U6FPEQw3uT/vf/AGpw0uG5yrToTn78f7v/ANseefaIP76UfaEk/jSvXP8AhsbS/wDoVdY/8DIf8a5b4tftdaVffCvxnCfC+rwhtJmBY3kPGeK4l4tUamlPDa/4v/tTs/1RxO9SX9f+BHyD+0d8err4T+PLDTI9Cs9TiutLjuhJNK6bD5k8f8H+5XlP/DZ14OT4P0wD2upv8a+zP2X9FPx88Fz/ABH0e6/sfT9C1toH0+8g8+ef7MkF0/lumE+fzNnSvc4f2zNImjE3/CKasd3z/wCutv8AGvyPNswwWOxtbGZjheSc/wC8fWYaljaFGFDL8Rzwh/dPyK+LH7QNx8VtCstMl0O10yO2uftPmQzSPv8Ak2fx10nhj9rK78M+G9J0YeF7G7jsLZLYTvcTAuE6cZ4r9VP+GxtK/wChS1b/AL/Wv+NH/DY2lf8AQpat/wB/rX/GuOWYZTOgsO4fu1/ekOGFzmFf26l7/wD26fl6f2zrvt4P03/wKm/xpP8Ahs69PTwdpp+lzN/jX6A/tC/tTaZ4t+APxH0iHw5qVnJeaBcwJJLNbeWnmfTmvDv+CePwjvvF3wTvPFdtq1tZ2+keKZpns3td8k3kQWk/yP2z938a2hh8qq4V4mjQ5rf3pB9bzSnW9jia3J/27E+KvjB8aJPi5/ZIm0m20v8As8zECGV38zfs/v8A+5XoUP7Yt7a2tvAfCNg/kwRwDN1MOEj2DvX6ix/tnaPcRiYeEtW+f5/9da/40r/tjaUI/wDkUtW/7/Wv+NY1MwymrThh50/ch/ekFPDZzTqTrwl78/8ACfl6v7Zt43/Mnab+FzN/jQf2zrpeD4Q03/wIm/xr6j8N/E+1/aC/ao8SeB9OtLnQb+6vNRm+332ydI/ITf8A6tP+udfS+i/FmP4G/ZfhzPo994n1HTf9Ze6d5EMc3nfv/wDVud/yb6ycMDT/AI+E5If42dP1jHT/AIOJ559Vy7H49fEb4wT/ABC8Y6V4gm0y2sH0+GKFbdGZ0fZIz5/8fr6V0f8Aa4ez+Geq6+nhLSbi+uHeOSD7ZJ+7y/p/4/X6Fap+1GNHg8+88C6zHH/z0+02sn9ao6f+2Ppsl9HBD4S1b98/l+X51r/jXZPHZZPkpzhpDb4jip4fNo89SEvj/wAJ+Z3h39u7VvDN0Hg8JWEkEn+uga8mAk/Wuuj/AG6fDuj77/SvA0H9pyLjyXu5sJ+Nfod4k/as0rw/qT2Q8MalJ8nmJ5c1rX5+fDXR3+M3/BRLxylnMulNqF9rd2n2pPP2fJL/AHK3w6yutCpGlR96H96RnWnmdKUJV6vJCf8AhPIfG/7WWoeMtA1zSJ/DVjbf2pE8ckyXExKb5PMPGea574VftBXPwu8NSaRFoVrqSS3j3RlmnkQ8oi7Pk/3P1r9atN+M1v8As/6fb/D7UdHutdvtJTfJqNl5EEE3nfv/AJEc7/49lWP+GxtKxj/hEtW/7/Wv+NcbzDLKdB4adPTtzSN1hs2qVvrFOV//AAH4T8wrf9tC8hmjdfB9gWR/Mx9pm/xr5tupTNcSSMuze27HSv1D/bD/AGxtHWHwbLL4W1KMf8TAIn2mEY/1H+xXuei6PJ+zAq+J9Vum8SW2tRQ6fHZ2MHkPE/lm58yR3P8AsbPyrfB46jgF7bC4W1Of2uYzxWHr4791iq376H2OU/ECjHvX7sf8NjaX/wBClqv/AH9tf8aP+GxtL/6FLVf+/tr/AI16H+s2FPN/1exv8p+FGPxpACtfuu37Y2lnp4T1Uf8AbW1/xr4e/wCCnvxisvi5Z/DA2uk3OmHTpNU3/anRt/mfZP7n/XOu3B57QxlaNGnuc2JybFYWl7aotD4NaNkwSpX8KZyzetft9o/h+4/ZgWHxXql8viW11G2g0uOx0218h1d40m373Oz/AJY/+P1s/wDDY2l/9Cnqv/f21/xrhfElKm7VocsvU61kVWrrhpc8D8KeW6CvY/hP+0Bc/C3wxNo0Wg22opLeNdGa4mkQ8oibPkP+x+tfqL8XP2vtKvvhX4yiPhfVYR/ZM4z5ttjnjt/hXnH7MegyftDeC7vx9pN7HoNjpOtNbPY30HnzzC2SC6Yo6fLz5mzpWOIzSOPoNRoc9Pr7x0YXATwE+epW9nP/AA8x8a2/7al5BNHMPB2m7kk3/wDH1Nj+deX/AAr+LEvwu8S6jrEGmW+pte2r2pgmd0RA7o+fk/3K/Y+L9s7R7uMTf8Ilq3775/8AXWv+NH/DY2l/9Clqv/f21/xrzqeYZTh4Tp06dlPf3pHbPB5ziJwqTl8H+E/L3/hs+7/6E7Tf/Amb/GkP7aF2P+ZO03/wJm/xr9RP+GxtK/6FLVv+/wBa/wCNee/tB/tS6Z4u+AfxH0aLw5qVk97oFzbpJJNbeWnmfTmsqVTJatRU1T/8mkb1P7dpw53P/wBJPz4u/wBsC9vrG8tR4S09PtMEkHmfapvkDxlCevo1cF8HfjRJ8JI9ZSHR7fVf7S8nInmdPL2b8Y2H/bNfaX/BPH4O33jj4H6h4ot9ZtdPh0vxVM8lpNbNI8/kQWk/3x0r6xh/bM0e6hEv/CJ6sfNG/wD11r/jXbVrZfgFXwk6PJB/3viOOEcwxzhiqNXnnD+78J+X/wDw2def9Cfpn/gVN/jR/wANm3x6eDdNP/bzPX6gt+2JpTR/8inqw/7b23+NfMXhz4m2n7QH7UvifwRptrcaFf3V5qF19uvtk6R+Qm/7if8AXP1rzo/2diE3hcLz2/vyPQ9pmlN/7ViOT/t2J8tn9s69X73g3TR/28zf415j8Tvi7N8SvFem63NpUFi9nCkIhgd3Rtkjv/H/AL9fsPpvxgtv2erK38Bajpd1r11piefJfWXkQwP50nn/AHHO/wDjq1/w2PpP/Qo6t/3+ta6MLj8rwc+eFPkn/ikceIw2b4yHLz88P+3T8v7j9tS8uJpJD4O03533/wDHzN/jTP8AhtC7/wChO03/AMCpv8a/UP8A4bG0r/oUtV/7+2v+NRyftkaVsf8A4pLVv+/1r/jXN9YyL/n3/wCTSOv2ee2+P/0k/MA/tnXfGPB+mk/9fM3+Necap8X5tS+Mdr8QG0uGOeCe1mFjHM+z9zGiAbuvPl/rX0r8NNEk+Mn/AAUW8cR2kq6S2oXut3UbXUPn7P3cn9yvuvTvi/bfs92MPgHUNNuteu9NTz5L6x8iGBvPk877jnf/AB1606mAyupzwoWhOPxc383Q8yLx+Yw5PbXnCXwcv/kx+Y3/AA2XeHp4S0/8bqb/ABpV/bMvWOB4N01j7XM3+NfqD/w2NpX/AEKWrf8Af61/xr5z/a6/bC0NU8GzT+G9Vj/4/Rj7Tb/9MP8AYNeZF5ViJcmGoc8/8cj1L5vTtLFVeSHflifRH7YX7zw34S/7Cc3/AKINfMdfTn7X3/It+Ev+wnN/6INfMdfI5p/vJ9Dk/wDusP66hRRRXj6nt6nzL+2sNzeCf+uF1/6Glfp9+2d/yB/Dn/YWuf8A0Qa/MT9tL73gr/rjdf8AoaV9L/t9ftp+D/Gng/wwnwn8exanqcOrTz3McOnTR7IXhCJ/r4R71+nYXBTx+UUKMP73/pR+aYrFRwObzrT/AK90K4v42f8AJHfGv/YP/wDbiOsXw38fvAknh3SH1bxhbDVfscH21ZLW6z53lpv+5Bs+/vpnjD4ieFPiT4O1zwr4Y12LWtf1W28izsYoJ0MsnmRv990jT7iPXyeGy7F0MZTlUhK0J78ulu/ofYV8dhq+FnCE43nA+jf+CdP/ACYr4s/6/wDXf/SKCuBt/wDj3T/cr1n9hXwnqvgj9i/xdpus2RsLv7XrMnll0f5DZR/3PpXk1v8A8e6f7ldfEk4zxPPA8rhuMqdOpCX9bjqKKK+S1PsNTi/jZ/yRnxp/2Dx/6Pjr3j/gmx/yZJ4g/wCwjrX/AKSQV4R8a/8AkjXjT/rwH/o+Otf9j39qT4b/AAp/Y98V+F9Z8awaJ40uH1qaxsZLK6kbfNaRpAfMSB05dK/ReH6c6+W1oQ/nPzziCfJmEJz/AJf/AJI1rP8A48YP9yOpq+fPgj+0RaXOk6qfH3ixIrpZoFshPayY8vZIJP8AUp/uV6Kfj98Nf+hvt/8AwEuv/jFfLYnJ8bhcQ6PJKa7qMrH1VDNMFiKXNz8hf/4J9Lj9vv4hbPv/AGTW/wD0qSvrb4i6ZBYfGDxTP9th/tK6S12W/wDc/wBFSvmj9gfwPrNj+2J4k8ZyWRbw9qmnapfWF8JUzNDNPG8b7Cd/zI/G8d69r/aQ8Pzx/FLXdet33wTG1d5I/wDWQP5CV9jnNaCoRlFc3wnyGV0Z/W5wl7pyNn4o1Tw/qs9lr2+8gf8AdzxyfvP+/dM1S3TwfqtpqmlTw3FpN5kkEclR6h4ktfEHh/ZqUL/2tD/qLiP+OuZr4WpWPu4USzqGoT6pfSXVzJ5lw9cR+wD/AMn7/EH/AK8td/8AShK66vIv2Uvi14T+DX7aXj7X/Getw+H9JaPWbRLqeGaT988/yJ+7R37H8q+p4b56jrr+6j5niS1OlTPqH9oJg3xm8UFembb/ANJYK87ryr42/tT6b4k/bD1a90fxgk/wsu3tcXH2KTY6JYxpJw8fnY85HrXb9oD4bqOfGNqP+3W6/wDjFeVmOTY6jWuoc6n/AC9DvyzM8LWw6TnyOH8x5H+2t97wT/1wuv8A0NK/UP8AbP8A+Rb0P/sNP/6Ikr80PjfpTfHq30e68BPH4jtdHSWK8lizB5TOd6DEwTrsf7npX6X/ALZTbvC/h0+utOf/ACBJXs1oujlFOhL44c3/AKUeRTjfOvbfYn/8ifK1FFFfn2p92FfM37a3/Mlf9cbr/wBDSvpmvmb9tb/mSv8Arjdf+hpX03Df/Izo/P8A9JPnuIP+RfP+vtH6g/tlE/8ACN+H/wDsNP8A+iJK+V6s/t7ftreBvGHw+0CH4WeP4NS1yDXHuLmKHTp4/wBx5Dpn9/CEPzn9a8o8N/tAeBJvDukPqvi+2GqfY4ftiyWt1nzvLTf9yDZ9/fXo5tkuKnP29KF/LqcGSZphVR9hUnyG58aTj4OeMz/04D/0ojr3b/gm6xH7EfiDH/QS1of+SsFfM/i/4heFviV4N1zwt4Y12LWvEOqW3kWljFBOhlfzI3++6Rp9xHr6z/YJ8K6p4H/Y98RabrVmbC++26zJ5ZZH+Q2sY/g9xXbldOWFy6rRxC5JuX9fI87N2q+MhXoPngeNWf8Ax4wf7kdTVDZ/8eMH+5HU1fnrvc/Q1sgri/jZ/wAkZ8a/9g//ANrx12lcX8bP+SM+Nf8AsHj/ANHx114H/eqP+KP/AKUcGN/3at/gke8f8E2f+TJ/EP8A2Fda/wDSW3rzWz/48YP9yOsn9j39qL4bfCn9jzxZ4X1nxtb6H4znk1qaxsZLK4kbfNaxpAfMSB05dP8APSvGPgh+0Na3Wk6qfH3ipIrpZoFshPbSH93skEn+oT/rnX3+fZXisVOdenD4PvZ8Pw/mFHDP2NQ+g68//YPUr/wUJ8Ws33Smvf8Aob05fj98NWbA8YW2f+vW6/8AjFdJ+w78P9dtf2y9V8YSWDDwzq1lq15ZXkkqAzQzjfC+zO/Dq6du9cWRUKuD9u8QuTnj9o7c+qwxcIexfPyfynr/AO0J/wAlo8UfW2/9JYK86r0X9oT/AJLR4o+tt/6SwV51XxuK/jzPrcH/ALvT/wAAUUUVyK9zpd7HA/sI/wDKQvxYf9jXv/Qnr6D/AGhP+Sz+KPrbf+ksFfKX7L/xU8K/B79tzxf4h8Y6zDoOjJNrUAupoZpP3ju4TiNHetX41ftUaZ4q/bA1e90jxgtx8Lrt7TF0bOTY6JYxpJw8fnY85Hr9VzbLq+PoR9l9mJ+ZZZjaODx03P7R6nXzR+2kWH/CF4+75N1n/vtK9Y/4aC+G2cf8Jja5/wCvW6/+MV5v8dNLf49W+kXHgKVPEVto8cyXskObcxM53oP32zf9x/uDtXzWRYbE4PHxxGJp8kNdZeh9Rm+Ip4rByo0Z88/7p+ln7Y3/ACKPhj/sKf8AtrJXi3wn/wBfq3/bD/2pXtH7YX/Io+GP+wmP/SWSvDPhfefZ9cng/wCe8H/oFc2Lt/aI8Av+E6x3fij/AJFzVtn/AD6v/wCi68j8N7JPEGm7/wDn6j/9GV7dcW6XEEkD/wCrdPLevBri3n0e+kg/1c9q/wD6BXPjP3c4TO3B/BOBB+15ePp/h+0uof8AWQ6XqUif+Q6+o/2wP3fgrwp/2E//AG0kr5c/aQuIPFmh+Htn7yO906+jeP8A79/u66H/AIKAftM6TrPw/wDDVh8MfFn9q6/b6t5k0Vjp8z/uPsvl/wDLSDYfn7D+/X3jws8wy2hTov8Am/8ASj4N4mGBzOc5/wBe6Gn+B9X1DTvtsKJ5b/vEjkf949cD8R9bn8M/DrxPqEf+strX54/76efH5kden/Dz4weHo/h14Z/4SDxFb2+tJplr9vjmgfKTbI9/mfJXhPxJ+JfhL4raH4q8N+EdYh13Xtc3w6fY28E6STP58b/fdI0+4j18nDKa+GxFGShOynr/AJn1zzGji6FaHPD4D7e/YxjS4/Zm8SI37yN9T1P/ANJY6+e/Ccb6X4j02D/lpDPHB/7Tr6V/Y/8ADOp+D/2c9d03WLX7Hd/b7+Ty96P8n2aP+5Xzr4fj/tDxxBs/1f2qSf8A9qV6Gf8AJUrU5wPH4f8A3dKpCZ33xEj/AOKRu/8Afj/9GV5FXqXxMvPs/h+OD/lpNPHXltfJY/8AjH12X/AcV8bv+SL+Nv8AsHR/+j4K9t/4Jkf8mX+M/wDsN61/6bbSvE/jd/yRjxt/2Do//R8FX/2Mf2nPh18I/wBkHx14Z1rxjDofjG6utYuNOtZLGeR3eSwgSB/kjdP9ZGfv+lfZ8P0518tqwh/P/kfG8QT5MwhOf8v/AMkdBZjNjB/uR1LXz58Ef2hbG70nVT4+8VpBdLLAlks9tIf3eyQSf6lP+udejf8AC/vhx/0OFt/4CXX/AMYr5XFZPjsPWnS9nzr/AAyPq6GaYKtT5ucf+wu2f+Chni4f9M9e/rXv/wC0L/yWfxL9bb/0lgryL9h3wJrf/DZuqeNG08nw5rFlq97Z3zSpmaCb543KE7/m3j747+1eu/tCf8ln8S/W2/8ASWCvoeIpRnh6Sj/dPm8lhOGOrc/9e8edUUUV8FqfeahXAfsM/wDKQ/xb/wBc9e/9nrv68a/Zl+KnhX4Oftw+MvEPjHWIvD+iq+swfapoZpP3juQiYjR3/wA/n9zwveo66X8qPjOJ/wCDT/xH1T+0E274zeKD05tv/SWCvO68l+OP7VWn+Jv2vtWvND8ZJcfC66nsSbn7G+x0SygSTh4/O++j111v8dPhrZWyXl54ts9jf6m3+yXXz/8AkP7ledmOT42nWvGHPz/gd+W5thp4dJz5HA8l/bDt7eC08G3V6D5ey9MMI/5av+4/8cr9Lv2x/wB54R8L/wDYU/8AbSSvzV+O0Mv7QqaVe+B508SR6OJhfyRZgFtv2GP/AF2zrsf7gx8lfpP+2E27wf4Xx/0FP/bSSvYrc1HKKdCXxx5tP+3jyKcfaZv7dfBP/wCRPlqiiivz/U+7Vwr5o/bU/wCPbwZ9b3/2jX0vXzR+2p/x7eC/re/+0a+l4b/5GdH5/wDpJ89n/wDyL639faP08/bE/wCRN8L/APYUH/pJJXy1U37eH7avgfxd8OvDkPwq8fw6lrdrqwluoodOnj2wfZdn/LeEJ9/0ryvw38f/AAPN4Z0l9V8YWw1d7OH7YJLW6z52z5/uQbPv16ObZLi5z+sUoX8up5mSZphY0PYVJ2N74yHHwh8a/wDYPH/oyOvZP+CZrEfsW+M8f9BzWv8A03WlfPXiz4jeE/iP4O1zwv4Z8QRatr2sWwtbOxignQyv5kb7N7pGn8FfUv8AwT88Iav4F/ZC8Yadrtn9gun1XVp44y6P8hsbcfwe6PXdlcJYXAVaOIXJNy/p+hx5w1XxlKtRfPA8ls/+PGD/AHI6mqGz/wCPGD/cjqavzx3ufoS2CuQ+Mn/JIfGv/YMH/oyOuvrj/jJ/ySHxr/2DB/6MjrswH+90/wDHE48V/utT/BI9m/4Jk/8AJlvjcf8AUb1r/wBN1pXB2f8Ax4wf7kdYH7Ff7Tfw9+Ev7IPjvwxrvjGHQ/F93d6vPp1o9pPI7+Zp0CQP+7jeP/WRn7/pXjvwP/aHhutP1oeP/FccVyjWqaf51tIcptk8z/UJ/wBc/v199n+VYrFc9alvHp3PgeH8wo4ZulU+2fQ9cN+w2GH/AAUK8Vkfe269j82qMftAfDh2wvjC1P0tbr/4xXSfsPeC9eh/bQ1Lxu1kf+EX1W21a9s71pkzNDN80b+X9/5/MT+DvXHkNGphPa/WocnNH7R6GfVaeKow9i+b/CfUHxI0u30/4weKJhew/wBpXS2vl2/9z/RUryWz8Uap4f1Wey17feQP+7njk/ef9+6679pDw7PD8Utd123fzIJvsrvJH/rIH8hK4rUPElr4g8P7NShf+1of9RcR/wAdeHiqnJXnD4D2cHTcsPTn8RJqlung/VbTVNKnhuLSbzJII5K57UNQn1S+kurmTzLh6rUV4kp+0Z7MY+zRwv7C3/KQvxX/ANcte/ma+gv2iv8AktXiL/ds/wD0lSvlT9mH4qeFPg5+294x8Q+MtZi8P6KkmtQC6nhmk/eO5CJiNHf9P/r6Hxy/as0zxZ+15qN7ovjGO7+F11JYkXH2KRF2JYxxyD54fO/1nmdu9fp2bZdXx+Hj7H7MD82yzG0cHjpuf2j1Ovmr9tAsLfwZjpm96f8AbCvVv+Ggfhv/ANDja/8AgLdf/GK82+OelyfHqz0ebwDLH4jh0YTC/khzB5O/Z5Y/fbN/3H+5XzWRYXE4PMI4jE0+SFnrL0PqM3xFPFYKdCjPnn/d/wAR+mX7YP8AyI+gf9heP/0lkr5Vr6q/a+/5ETQP+wvH/wCkslfKtePnH+9HZk2mECiiivG1Pb1Pmj9tT/j38F/W9/8AaFfqF+1//wAiL4dP/UXT/wBJZ6/L39tT/j38F/715/7Qr6m/bt/bU8B+LvhTodt8K/iBDqXiGz16O4nhh06eP9x9mmQv+/hCff2dP79fp2FwU8flFCjD+9/6UfnGKxUMLm860/7v/pJQrj/jJ/ySHxr/ANg8f+jI6wPDf7QHgaTwzpMmq+MLYau9nD9sEtrdZ87Z8/3INn36b4t+I3hP4i+D9b8L+GvEEWra9rNsLWzsYoJ0Mr+ZG+ze6Rp/BXyWHy/F4fF0+enO0J78unr6H2FfHYbEUJ8k4/AfRv8AwS9/5M68Wf8AYwav/wCm60rz3T/+PG0/64JXr3/BO/whrHgL9lTxRp+u2f2C5fW9VmjjLo/yfYbcfwH1jevIdP8A+PG0/wCuCV6HEkozrc0DxuG4yhTqRmTUUUV8dqfX6nHfGb/kkHjb/sH/APtSOvbf+CWv/JnvjH/sYtW/9N1pXiXxl/5JD42/7B//ALUjrT/YX/ae+Gvwe/ZW8YeHPEfjK28P+J7rUdUurK0ktLqR232MEcD/ALuN0/1kZ6+lfovD9OdTLq0Ifzn59xBPkx1Oc/5TZ04ZsbT/AK4JUtfPPwQ/aDtZtP1cePvFaRXEbWsen+dayHKbJPM/1Cf9c/v16R/wv/4cDj/hMbX/AMBbr/4xXy2KyfHYevOl7PnXflkfVUM0wWIp83PyEn7DLZ/4KGeLB/s69/WvoL9or/ks3iL6Wf8A6SpXkH7DvgXXYf2zNU8ZHTyfDesWWr31jfNKmZoZvmjcoTv+bfH98d69f/aK/wCSzeIvpZ/+kqV9DxFKMsPSUf7p81ksJwx1Xn/r3keb0UUV8Fqfe6hXnP7FP/KRTWf+u/iH/wBFyV6NXiP7OHxQ8MfCP9unXvEXi/WItE0eC61qBryeCSTY7pIif6tHfqa+74Xu3XS/lR8bxP8Awaf+I+tv2imDfGbxCV6bbL/0lSvN68p+On7Vmm+Kv2v9QvNE8YxXfwsupLEi4+xSIuxLGOOT78Pnf6zzO3ethv2gPhuvXxjaj/t1uv8A4xXmZjk2Oo1rqHOp/wAvT1O7Kc0w08Ok58jh/MeT/tpf8e/go/7N7/7Rr9Of2sOPhr4P/wCv+H/0hkr81fjdpx+Plpo83gSSPxHDownN/JCDB5O/Zs/12zeTsf7g/gr9Kv2szt+GvhJG+/HqEMf/AJIvXsVYujktOhU0nDm/9KPGjH2mde3XwT/+RPGPhP8A6/Vv+2H/ALUrsPFH/Iuats/59X/9F1wnwvvPs+uTwf8APeD/ANAr024t0uIJIH/1bp5b14uD/eUD3MT+7rniPhvZJ4g03f8A8/Uf/oys39sC8fT/AAzaXUP+sh07UXT/AL9x1PcW8+j30kH+rntX/wDQKg/aQuIPFnhzQtn7yO9sb5Hj/wC/f7uvR4a/33kPP4i/3LnPp/8Aaq/5Jb4NP/T7B/6QyV4Dp/gfV9Q077bCieW/7xI5H/ePS/t2/tK6Jf8Awj8NaT8NfFf9reJ7XVoJJorHT53H2VLJ45P9ZBs++U6f36v/AA7+MHh+P4c+Gf8AhIfEVvb62mmQ/wBoRywPlJtnz+Z8ldmOyStUn9YnE4MtzejToewczzH4h6vP4d+HvijUE/1kNl88f99PMj8yOvrj9h2NLj9nvxTG/wC8jfWNR/8ASWCviD4i/E7wn8UtF8VeH/CmuQ6xrOuK1rp9jFBMnnP5iP8AfdETsa+8P2L/AAzqng/4Fa9pusWv2O6/tO9k8vej70+ywf3K68opVMNg61Gv7nvfgcmcVKeIxVOtT/lPmXwnG+l+INJg/jhnjg/9p16b8RI/+KRu/wDfj/8ARlcD4fj/ALU8aQbP9X9qkn/9qV2fxMvPs/h+OD/lpNPHXxtL3KNQ+xq+/Vpnltcd8Zv+SQeNv+wf/wC1I67GuO+M3/JH/G3/AGD/AP2pHXBgP97p/wCOJ6GK/wB1n/gkez/8Es/+TPvG3/Yx6t/6a7SuD0//AI8bT/rglYv7CP7Tnw4+EP7LHjbw14l8YRaD4ovNU1S6sbV7WeWR9+nQRxv8kbp9+Nx8/pXjHwN/aCtrjT9YXx94qjiuY2tU0/zrWQ5j2SeZ/qE/65/fr9A4gyvE4vnr0o6w+9+h8Dw/mFHDN0an2z6Hrzj9ilSv/BRHXCenn+If/RctWR+0B8N2bA8Y2uf+vW6/+MVs/sV+AdfH7aZ8aNZE+GNUTV72z1Bpk/ewzQyeXJ5f3/n8xP4O9cWQ0amD9q8UuTnj9o9DPqkMXRh7H3+T+U9t/aM/5LP4i/3bL/0lgrzevSP2jP8Aks/iL/dsv/SWCvN6+LxX8ef+M+ty/wD3Sn/gCiiiuTW50u9jzf8AYo/5SN60f+m2v/8AouevpD9or/ktXiLP92z/APSVK+S/2cvij4Y+EP7dHiDxH4w1dNE0a3utaha8khkkwzpKiD5Ed+9bPxz/AGrtN8Wftfaje6L4yS8+F909jtuvsciJtSyjST78fnY87zPzr9WzbLq+YYWPsfswPzHLMbRwOOm6n2j1OvAv2wbSC30nwbc3uRk6h5MI/wCWr/uP/HK9Rt/jp8NbO3W8vPFtntbiG3+yXXz/APkP7leW/HhZP2grLR7nwLPH4jGj/aXvzHmAW3nbPL4mCdRC/wBwfwV8zkeFxOExsa+Ig4QXWX+E+mzfEU8Vgp0cNPnmfpD+1fJ5nw38IP8A9RCD/wBIZK+XK+pP2r4/L+G/hCN/vpqECf8AkjJXy3Xl5x/vR15HpgoBRRRXhanv6nzZ+2if9D8Ff799/wC0K/TP9q7/AJJh4QP/AFEYf/SGSvzM/bW/48/BP+/ff+0K+m/23v20PA/ib4M+GLT4X/ECHUPEVrq8Etzbw2E8e2D7E8cn+vhCY37On9+v0/C4OeMyahQh/e/9KPzbF4qGEzmdaf8Ad/8ASSvXM/FDH/Cq/GWf+gVN/OuX8L/tAeB5vC+ivrHi+2GsPZp9sWS1nyJv4/uQbKf4i+J3g/x14a1nw34f1+LV9d1izeys7GKCdDNM5+RN7pGlfHUctxeHxNPnhO0J/wAulu/ofY1cdhsRh58k4XPoH/glnz+x742/7GTV/wD012lcHp//ACDrT/rhHXrn/BObwbq/w/8A2V/Gena5Zf2ffPr+pzpHvR/kOnWn9wn+5Xken/8AIOtP+uEdepxLKE6/NA8ThuMoU6kZ/wBbk1FFFfG6n2Ry/wAVP+SU+NP+wTNXr3/BLHj9j/xt/wBjFq3/AKa7SvIfin/ySnxp/wBgmarH7CH7Tnw4+EH7LPjbw34l8YRaD4ovNU1S6sbWS0nld9+nQRxv8kbp9+Nx8/pX6Jw9TlUy+tCH8/8AkfnvEU+TG0Zz/kNvT1P2G0/64pUtfOvwN/aHgurPWk8f+K44ZE+zR6f51tIcoPM8z/UJ/wBc/v16Z/wvz4bjj/hMbbP/AF63X/xivl8Zk+Ow9b2Xs+df3YyPq8Pm+CxFPn5+QT9jX/lI5quz7/ma/wD+iJK+yfiXpcGn/GDxLP8AbYf7Sura2jS3/ufuUr5T/Yv8E6z/AMNvL48hsvM8LakmsXlnqDTJmWGSGTY+z7/p/B3r6A/aY0O4i+KWta9bv5kEyWv7yP8A1kL+QlfYZxVh9UpTiubk5T4/LqU/r1aEvc5zjrPxRqnh/VZ7LXt95A/7ueOT95/37pmqW6eD9VtNU0qeG4tJvMkgjkqPUPElr4g8P7NShf8AtaH/AFFxH/HXM18NUrn3dOgWdQ1CfVLqe6uX8yd681/Y2H/GyLWf97XP/RMleg14t+z98UPDHwi/bz8R+IvGWsx6Jo1vPq8D3kkEkmHdJEQfIjv3r6ThfnqVa/8AhPmeJPcoQ/xH1h+0Uwb40+ISvTyrP/0lSvOK8m+PX7Vln4o/a6vrzQfGMd18Lbq407/SI7IonlpZQJP9+Pzh84krbb9oD4br18Y2o/7dbr/4xXBmOTY6jWuoc6n/AC9PU78pzbC1cOk58jh/MeU/tp/8efgn/fvv/aFfpn+1euPhn4PP/T5D/wCkMlfm18cLUfHrTtGPgSaPxLFohuX1CSLMHk+ds8sfvtm8kQv9wfwV+lX7WC7fhj4Pjb78eoQx/wDki9exVi6OS06FTScObT/t48aL9rnXt4fBP/5E+W6KKK/PtT73UK+bP20v+PPwT/10vv8A2hX0nXzb+2r/AMefgn/fvv8A2hX03Df/ACM6Xo//AEg+ez//AJFtX+vtH6bftWD/AItd4VP/AFELX/0hnr5bq5+23+2l4D8UfBXw/p/ww8e2+peJrTV7WSeK3sZ0/wBFSymjk/18IT75TpXkvhb9oDwPN4X0V9X8X2w1j7HD9sWS1nyJv4/uQbK9HNsoxc5/WKUL+XU8vJM0wsaPsKk7HT/FI4+FfjM/9Qmb+dex/wDBKlj/AMMj+Lz/ANTRqn/pusa+ePEPxO8H+PvCuteG/D3iCPV9c1izeytLGKCdDLM5+RN7pGlfUf8AwTd8G6t4A/Zh8W6brdmbC+fxFqE6IXR/kOnWv9wn+5XdlcJ4bAVqOIXJNyOTOWq+LhXoPmgeQ6f/AMg60/64R1NUOn/8g60/64R1NX5473P0JbBXM/FD/klvjL/sFTfzrpq5n4of8ks8Zf8AYKm/nXZg/wDeKf8AjicWM/3eoexf8Eqf+TR/F/8A2NGqf+m6xrznT/8AkG2n/XBKo/8ABP8A/ag+GfwZ/Zn8V+HfFvjK38P+I7rWdQurWzktbqfej2VrHHJ+7jdPvxv19K8N+Bf7QkN1Z62vj/xVHFLH9lj0/wA62kOUHmeZ/qE/3Pv1+gcQZXicW516Ub8n3v0Pg8gzGjhm6NT7Z9FV5/8AsaZ/4eN6xnp5mvf+iZad/wAL8+G7NgeMLbP/AF63X/xitz9jXwPrkv7bDeN0sd/hTVYtXvbPUGdAZYZoZPLfZ9/qf7lcGQ0amDdR14cnNH7R6efVKeKow9jPn/wnt37Sn/JaNa/69rL/ANEpXmdemftKf8lo1r/r2sv/AESleZ18bjv96mfU4D/dKf8AgCiiiuM6uhwH7G//ACki1o/7Wt/+iJK+iv2kv+Sza3/172f/AKJSvk/9n34oeFvhH+3l4j8ReMdYTRdEt7jV4GvJ4JZPndJEQYRHfuK0/j9+1fpviv8Aa0lu9A8Zx3fwyuDpyzXKWUiL5aWsaTj54fO+/wCZ+dfquY5bXzDCRVH7MIn5hl+Po4HMZuf2/wD5I9Sr5t/bQz9h8GH+Hfe5/wDIFep/8NBfDbkf8Jja/wDgLdf/ABivOvjhYv8AH7T9E/4V9MviaPSXn/tDbmBYmm8vy/8AXbOX8l/uf3K+byPC4nCZhHEYim4U11l/hPqc3xFDF4OdGjPnn/dP0g/a+/5Fvwl/2E5v/RBr5jr6j/a+sLn/AIRvwsPJfP8Aac3/AKINfLleNmqaxOp3ZM08LC3mFFFFePqeyfM/7an3vBP/AFxuv/Q0r7p8R/8ABMn4AaC4Gq6p4j0+33+Wkl94iggR3/2N8FfC/wC2tG8c3gz5P+WN1/6Glfp3+2nb/wDEj8Mb0/5i83/og1+nYbFTwWTUJw/vf+lH5nisLDGZzOjL+vdPGP8Ah3j+zL/0Neq/+FZZf/G65T4rfsofBP4B/DPxH8QfAniG5vPFfh+3F1p8F5r9pfQvJ5iR/PCkfz/I71l/Z0/uJ/3xXF/GaFF+D/jMhE4sAeE/6eI68zD8QVsRiKdCfwTPTr8P0sPTnXhU+A+mP2J/HF98Rf2N/Fmqamlsk32vWoP9Fh8uPYLWM/8AtSvG7f8A490/3K9C/wCCcsLt+wr4sCo8n+n61/6RQV57HG8cEe9PL+SuPiOnCjXjGCOjhupKdOpKY6iiivkEfY6nGfGv/kjXjT/rwH/o+OrP7Ev7FHw1+PfwFk8XeL5tchvo9XvbNpLDU47WBII44H53xv8A33qH42Rv/wAKZ8avs/5h/wD7cR17v/wTYt3uP2K9cgRN8j6rrSIn/brBX6Nw/OdLLas4fzn55xDGFTH0YS/k/wDkiiv/AATv/ZiK/L4r1L/wrLL/AON11Ph39hn9nfwfaLfWetXonun2faJ/FMAH/ffl155qHw7n0Pw59tmntZJIUj3xxpU/h/WNLuNH/sTW08uCN/MguI0+5XPDiDExqclQ3/1foez54P8Ar7y/+yZ8Yv8AhKP2vPGnw4+y6emmeFoNW03R9St4jJcTWttPHBB5k2/5/wB2n/A66b4yeIr3wz8Z/FkLQpcWk5td8En+rf8A0VPnryT9iPw74Z0n9tTxx/ZeqPq+q3FjrbMkbY8n/SEr0n9oi3uY/jP4l83fhDa/f/69YK6OIFCjQhUpo58jfNi506jPNqKKK/Odz9E1Fjke3kjkSTy5Erxv9nz4GeEf2kf2uvHfh7xMt9Bbr/a9+yabdeRIJkuE2OjbH/vv8nsK9kjjeT7ieZXJf8E/1aP9vjx/hW/489dC/Xz0r7vhiXs3XlH+VHxnE0bxpo9t1T/gnH+zfp181tf+IdZs7mP78F14otUkT/gLwVH/AMO8/wBmX/obdU/8K6y/+N1pftDaXHefF7xQ8aZukNr50b/x/wCiwfPHXl32dP7if98VNbiHE05zh2MMPw/QxFOFS+/9dzgv2rNC8J/sb2/hqw+FGoQ6za+JI7qbU/7Qv49T2PBsSPZsCbP9ZJz3/wCAV9zftkoIPC3h7H/Qacf+QJK/L/8AbQjSM+DQiJH+5uv/AENK/Uf9s6xmbw5oeIXI/tpj/wCQJK9HGTjiMqjiuT35/wCZyYaEsPmiwzn7kD5Rooor801P0UK+Zv21uvgn/rjdf+hpX0zXzT+2tG/m+Dfk/wCWN1/6GlfTcN3/ALTpP1/9JPneIP8AcJr+viPubXv+CZH7P+gtnVNU8R6fb79iSX3iOCFHf/gcFZf/AA7x/Zk/6GvVv/Css/8A43Xt/wC2dbj/AIRnw/uTP/E6b/0RJXyl9nT+4n/fFe5js9xOFrezPAwORUMXQ9vc0/it+yj8E/gH8M/EfxC8CeIZ7zxX4fthdafb3mv2t9C7+ZGnzwJH8/yO9esfsO+N774ifsh+JdY1RLeOc32swf6LD5cez7LGa+X/AI0Qovwd8aEImRp+fuf9PEdfQH/BNG0e6/Yr14J9waprJ3/9usH+Fd+GxX9p4OeKnD34e6cWMwry3EQwsZ+58R5vo9m95BAifu40gjkeST+CpLjyPPk8nf5f/TSpJLjy7GC1hR44ESP/AFn+sf8A6aVWr8zlo7M/TYu6TCuL+Nn/ACRnxr/2D/8A2vHXaVx3xvjf/hS/jZ9n/Ll/7XjrowP+9Uf8Uf8A0o4sa19Wrf4JE37Ef7E3w2+P3wFl8XeLp9bh1CPV72zaSx1JLW38iOOBxw8D/wB969kj/wCCd/7MWfk8V6l+Hiyy/wDjdXf+CbqsP2J/Eg2tn+1da/h/6dYK8vs7dPstv8ifcj/gr9FzrOK2BrclM/PcpymljqTnI9J/4d4/szA5Hi/VAf8Asb7L/wCN1wf7IPj+5j/bZ1/4b2P2SXwv4aXWtM02eOMPM9ranyYPMkH3/kjT56i+zp/cT/viuB/YNVn/AOChXi0IvOzXf/Q3qMqx39rwq+3h8BvmOXPKYQ9lP4z6G/aF/wCS0eKPra/+ksFec16T+0TZzR/GjxTvR+tr/wCksFebV+fYvSvO599g9cJTt/IFFFLHG8n3ErjV7o6XseBfAH4J+G/j9+2f4w8K+KBff2bJNrN2Dp10IZN8bu6/Psf/AD619e3/APwTj/Zv0++e11DxDrNncx/fguvE9qkif8BeCvA/2DVY/wDBQjxV8rfd17/0M173+0Nbg/GrxTuRC+bb+D/p1gr9XzTMa2X0qU6f90/LcBl9LMMVVpzGf8O8v2ZR/wAzbqn/AIV9l/8AG68E/au0Hwh+xva+GtP+FGoxavZeJY7qbVP7R1CPU9jwGNINmwJs/wBZJ/lK7z7On9xP++K+Z/2z40jXwYERY/3Nz0/30ry8tzipmuLjhcTD3JX/AAR6mMyiOVUZ4mjU96J+n37YX/Io+GP+wp/7ayV8yaXqD6PqUF6n+shfzK+o/wBsOzm/4Q3wx+5f/kKf+2klfKdfMZpLlxR9Jk7VXBHvVneJeWsc0L+ZG6eYlcZ8QPCb3n/E0s08y4RP38cf8f8A00rC8F+MP7Dk+xXn/Hg/3JP+eFeqW8iXEcbwukkb/ckjrphOGKhYy5J4SZ8FftQeOdW8Dah4NutLmXZNDdb4Jv3kb/OlfoPrn7Ivwk0KOKTU77WtPjc/LJd+I0hjkfZv/jSvlD9s74Y6T4qk8PT3FtPb3aWt1Iklon/TRP8AWV9x/tReCf8AhIfDPhyN7ry401DzP9T5n/Lqa+yo1K2Ayyj7H+9/6UfGV6dHGZnNT/r3TyXWv2TfgR4isHspvGWpWlu/3/s/i+2+dP8Av3XE65+xn8Gfgv4Z1b4jeCdXvdQ8QaDB9u0+O712C+gefzNnzxpHHv8Av/362tL+FekWcm+5R7yT/pp/q6g+NGqaf4Y+D3iy5uNlvaw2X9z/AKaR/wCrrkwuaYmvU9nW+2dFbK8PQp+0ov4D2P8AZv8AHGoeJv2c/Fmp3/2WO8ju9TgT7OmyP93ax14b8H9H/wBFk1R/7kcCf+1K6b9g3xNN46/ZR8baoLbyoV1nWY4I/wDpmbG3xv8Azrkv+Esh0Pwdpul6bv8Atf2WPfJs8vZ/z0rizpRhiL/yHbkXNWoVF/OVPiBrn9sa55EL/uLL93/20/5aVzNFFfDzn7SfOfdU6fs4chxfxt/5Ix42/wCwdH/6PgqT9hn9iz4a/tAfAnUfFfjCfW4b+2166sN9hqUdrAlvHBBJ8++N/wDno9J8bo3/AOFK+Nn2f8uUf/o+Ovaf+CZcbf8ADF/jb5WB/tzWv4f+odaV99kFSdLLas4ae/8A5HwPEMYVMfRhL+T/AOSFT/gnd+zF/B4r1L/wrLP/AON0/wD4d5/szf8AQ3ap/wCFdZ//ABuvObO3T7Lb/In3I/4KsfZ0/uJ/3xXl/wCs2Juel/qzRtdP+vvJv2SPHlxZ/tt6/wDDawNnceFvDcWsaZplxGge4ktbU7IN8/8AH8kaV6D+0INvxm8Sj3tf/SWCvA/2F1Mn/BQrxYETny9e/ma9+/aGs54/jN4p3o/W2/8ASWCvR4hpxp4aE4Q+I4sjlKWMnCc+bkPOKKKK/Oz77UK8H+AvwS8OfH79tTxj4U8Ui/8A7Nkm1a7B0268mTfHIXT59j8V7zHG8n3E8yuP/Yc02W2/b+8UXM6sizR695KlDiXk5b/c5r7zhifK67h/KfGcTa0YL+8e63v/AATd/Zv0W+8u/wBb1e3vF+f7Jf8Aii2hf/0Rvpl5/wAE/wD9mu+m8+Xxjqskn/Y32Z/9p0ftE24Pxo8UGVMvm2/9JYK86+zp/cT/AL4qa/EeJpznT7GGH4coVqcKl9/67nDftX+GPB37Gen+GLb4TajDrEPigXcmqf2lqEGp7PI8uODZsCbP9fP9f+AV9v8A7YEYTwf4Yx/0FP8A21kr8xP2zo0jtvBYRFj4vOn/AGxr9Qv2wrSf/hD/AAx+5fH9pj/0lkr0cZOOIyqOK5Pfn/mcmGhLD5osG5+5A+VKKKK/NdT9FCvmj9tP/j38F/W9/wDaNfS9fNf7acLrD4M+THF5/wC0a+l4b/5GdJ+v/pJ87n//ACLav9faPuTWv+CZv7P+ixpJqmq+I9Pgf7kl94jghR32b/44Ky/+Hdv7Mn/Q1al/4Vln/wDG69o/bGtx/wAId4Vdkyn9qf8AtrJXyz9nT+4n/fFe5js+xOFr+zPAwORUMXQ9vext/Er9kz4HfAv4c+JfH/gnxLdXfinw/ZPqNhBeeIbW7R5s7PnhSP5/v16L+wb481H4jfsoeMdW1RLeOePVNWt0+yw+Wmz7DBJ/OSvnH4wW6f8ACofGPyJ/yDx/B/00jr2v/gmTA8v7FXjPYjuf7d1Y/wDlOtK7sLi/7Twc8TOHvx9048XhXluJhh4z9yR59Z/8eMH+5HU1Nt43jtIN6eX8kdOr8xd72P09bBXH/GT/AJJD41/7Bf8A7UjrsK5H4xRv/wAKb8bPs/5h/wD7UjrswN/rVH/HE4sW19Vn/gkN/YW/Yv8Ahr+0B8C9T8WeMZtch1C1166sPMsNSjtYPIjt4JPn3xv/AM9Hr2iP/gnf+zFt+XxXqX4eK7P/AON0v/BMWCW4/Y18aQLEzyPr+rRp/wCC60rF1D4dz6H4c+2zT2skkKR7440r9IzbNMThK1qex+b5VllDGU/ePRfD37DH7PHguxW/s9avBcXT7PPn8UwAf99+XXFfsv8Axij1n9sLxF8Mo7XT/wCyPDKavpuj6lADJcTWtsfLj8yff8/yJVbw/qml3mj/ANia2nl2m/zILiNPuVyP7H/hvwvo/wC3F4mOnaodT1eaHW3CxvxDz/sfjRl+NWZqp7T7A8wy95b8DPZPjN4hvfDHxo8WQtClxaTfZd8En+rf/RU+evI69K/aKt7mP40eJfN3/IbX/Wf9eqV5rX53jpuVecGfoeAjehTmv5ApY5Ht5I5Ek8uRKSljjeT7ieZXnpO6O17ani/wT+CvhT9oz9srxf4c8SLeQ2Ym1e7lTTbnyJA0MnyPG+x/U/JX1XqX/BN/9nLT757W/wDEOtWdzH9+C68UWqSJ/wABeCvA/wBhZW/4eGeLPlb/AFWvfzNfRX7RWlx3nxe8ROi5u0itt8bp9/8A0VPnjr9XzPMK2X4elOn/AHT8tweX0sfi6tORlf8ADvP9mX/obdU/8K+y/wDjdeD/ALVvhfwd+xpp/hi2+E+oxavD4oF2+qf2lqEGp7PI8tINmwJ5fE8n+Urufs6f3E/74r5r/bPjSOz8HbEWPm86f9sa8vLc3qZri44XEw9yd/w1PUxmURy2hPE0anvwP1C/bA/5EXw//wBhaP8A9JJK+U6+sv2wrOb/AIQTw/8AuX/5C8f/AKSyV8m18pnEeXFH0mRy5sEFFFFeNqe2fNH7af8Ax7+C/ref+0K+59a/4Jm/ADRokl1TVfEenwOVjSS+8RQQxu+zf/HBXw3+2pbtHD4MBXHF4f8A0TX6ffthKP8AhAPDbOuU/tdP/SWSv07DYqeDyWhOH97/ANKPzfFYWGMzadGf93/0k8S/4d5fsy/9DXqX/hW2X/xuuc+JX7JfwP8AgT8OvEnj7wT4lurzxT4fsn1CwgvPENrdo82dnzwpH8/36w/s6f3E/wC+K5L4vxpF8IfGnyIP+JYP4P8AppHXk4biCtiK1OhU+2ejU4fp4enOvCfwH0r/AME//HmpfEf9l3xTqmqJbxzx6zqdun2WHy02fYoJP/aleNaHZveWsCJ+7jSCOR5JP4K9E/4Ja2zz/se+KAq/L/b+rfP9bG0rg/tHl6baWsKPHbokf+s/1j/9NKx4ioxo14xgi+HqkqyqSkR3HkefJ5O/y/8AppUVFFfGn2mpx/xk/wCSQ+Nf+weP/RkdH7Cv7Ffw3/aE+Bep+K/F02uQ39rrt1YD+zdSjtYPISCCTnfG/wDz0enfGKN/+FN+Nn2f8w//ANqR17V/wS4jb/hj3xh8rZ/t/Vv4P+odaV+i5BUnRy2tUh/Ofn3EEYVMbRhL+UjT/gnd+zDs+TxXqX/hWWf/AMbqT/h3n+zN/wBDdqn/AIV1n/8AG6850+3T7DafIn+oj/gqx9nT+4n/AHxXmf6zYm56P+rNG17/ANfeS/sj+PLu1/ba8Q/Dew+xz+FfDcWt6XplwiB55LW1OyDfOPv/ACRpXov7RfHxm8Qj/Zsv/SVK8A/YXUyf8FCvFYROfL17+Zr6F/aOtJl+M3iLej9LP/0lSvQ4jpxp4aE4Q+M4sjlKWPnCpPm5DzGiiivz3U++1Cvn/wCCPwY8PfHb9tfxH4S8THUP7Kub7WZ2/s668mTfD5kifPsfv/n1+gY43k+4nmV5t+xSjL/wUU1glW/12vfw/wDTOSvvOGJezdd/3UfGcTa0qa/vH0Zqv/BN/wDZy02+a2vvEOtWdyn34LrxRarIn/AHgqv/AMO8/wBmb/obtU/8K6z/APjdW/2jrdP+F1eId6Iflsv4P+nVK82+zp/cT/viorcSYmlOcOxhQ4foVaEKl9/67nDftYeGfB37G+neGLb4TajDq8PigXr6oNS1CDU9nkeXHBs8sJ5f+vn/AMpX2/8AtZIG+GvhSTvJqMEn/kjIa/MT9tCOOO18FhEWPi86f9sa/Tz9q61nl+GvhH9y+Pt8H/pDJXo4yccRlUcaoe/M4sPGWHzRYZz9yB8x6XqD6PqUF6n+shfzK9xs7xLy1jmhfzI3TzErwWuq8F+MP7Dk+xXn/Hg/3JP+eFfEYKt7P3Jn22Mo+0/eQN34geE3vP8AiaWaeZcIn7+OP+P/AKaV8g/tOeOdW8Cz+EbjS5l2TLeh4Zv3kb58ivvW3kS4jjeF0kjf7kkdfMf7Z3wx0nxNB4enuLWa3u1gvpEktI/4/wB3/rK+tyjDWzOFeH9e6fL5nW/2CdOf9e8fVd/+yT8JNL06zu9Wu9Z0qK5RNr3niNIU3um/y/nSud1n9k34Ea/p72c/jLUbS3f7/wBn8X23zp/37r1f9o/wO2ufD/wvA115aQ3sEn+p8z/l1kSvC9L+FekWcm+5R7yT/pp/q69HFZpj6c+Q8jC5XhMRR9pNmTq37F3wX+D/AIY1f4i+DNXv9Q1zw9avqWnx3evwXUDzpzHvRI03/f8A79e5fsu+OdQ8UfAfxZql/wDZY7yPUdQgT7OmyP5LWB68g+MV9p/hn4ReK7m42QWsOnuB8n/XP/V1q/8ABPPxNN4+/Zh8damtt5cCeItUhhj/AOmf9nWld9Os8ThJ15/Eclej9Urwow+E5T4N6P8A6D/aL/8APCOBP/alUPiBrn9sa55EL/uLL93/ANtP+WlW4/FkGh+C9N0vTd/n/ZY99xs8vZ+7/eVx1fmeIrQp0vYxP0ylCdSrKtMK474zf8kf8bf9g/8A9qR12Ncj8Yo3/wCFN+Nn2f8AMO/9qR1z4G/1qj/jiaYpr6rP/BIi/YM/Yv8Ahx+0N8D9Y8V+LptchvrXxBdaaDpmoR2sHkJa2sn7zfG/ed69rT/gnf8Asxbfk8V6l+Hiyz/+N07/AIJaxN/wx/41+X/mY9VH/lOs6870+3T7DafIn+oj/gr9GzrOK2BrclM/OsnymljqTlI9I/4d5/szZ/5GvVM/9jfZf/G686/ZS+IF3p/7cV/8M7H7BP4U8PNrek6fcbA88lrapIkG+b+P5I0qX7On9xP++K86/YlXzP8AgolraIvPn+If/QJaWVZh/a8aka8PgN8xy55TCE6U/jPo/wDaM/5LP4i/3bL/ANJYK83r0z9pC0mT40eIdyP92y/9JUrzOvzrFq1ed/5z77L3fCU/8AUUUscbyfcTzK5Fe50vY+f/AII/Bjw/8ev22fEPhDxP9t/sma91i4b+zbryJN8PmOnz7H9K+xrr/gm7+zlo1/5Wp65rVndr8/2S/wDEtrC//ojfXz/+xXpslv8A8FCdWurhWUTTa8IU2n978knz/wC5z9+voD9o5AfjV4iLJl8Wf8H/AE6pX6vmmY1sBh6U6f8AdPy3B5fHH4utTmMvP+Cf37Nl9MZpvGOqyP8A9jfZH/2nXgH7WHhHwZ+xvpfhdPhNqMesQ+KjejV11K/h1PZ9m8jyChjCbP8AXzV3P2dP7if98V83ftnRxx2HgzYuP3l7/wC0K8vLc3qZlio4XEw9yd/w1PUxmTxy7DTxNKp70D9N/wBq+TzPhv4Tkf8Aj1CB/wDyRkr5br6n/ars5j8M/CH7l/8AkIQf+kMlfLFfMZwuXEn0GRvmwcAooorwtT3z5s/bW/48vBP+/ff+0K+3b/8A4Jo/s/6fptpd6rqPiPS7eZI9kl94jggTe6b/AC/ngr4m/bUt3+yeCvkb799/7Qr9L/2rkH/CrfCAdfk/tGH/ANIZK/TcNi6mDyajOH97/wBKPzbHYWGMzidGX93/ANJPHP8Ah3f+zL28V6l/4Vll/wDG6wviB+x38Cfg38PvE3jvwh4lurzxR4csJtU02G68Q2t2jzp9zdGkfz/Sud+zp/cT/viua+KEKL8L/GZ2IMaTMeE968rD8QVsRVp0Kn2z1anD9PD03UhU+A+h/wDgn34+1L4lfsw+MtY1SK2S5XW9Tth9lj8tNn9n2r/+z14zp/8AyDrT/rhHXoX/AASvheT9j/xtsR5P+Km1Mf8AlOtK89s7d7extN6eX+4jrPiOnGjXjGCNOG5yrU6kpk1FFFfFq9z7DU5f4qf8kp8af9gmaqf7B37GPw4/aF+COseKvGMuuQX1r4gutNQ6bfx2sCwJa2sn7zfG/OZ3q/8AFKOT/hUvjSTZhP7Jmr2D/glbbyy/sh+M40X94/iXU0/8p1pX6Pw/OdLL604785+fcQxhUxtFS/lGp/wTv/ZiKfL4r1L/AMKyy/8AjddToP7Dv7PHgux/tKz1q9E8/wByefxTAB/335deeXnw7n0PwzHezT2snkpHvjjSp9D1TS7zQ5NE1tPLtP8AWQXEafcrlhxBiYVOSZt/q/Q9nzwf9feTfs3/ABahvv2zNY+F1rBp/wDZPhs6vpukalH++uJoIE2R759/7x+K7b42eIr3w18ZvEsLQpc2k8NqHt5P9W/+ip89eI/sp6D4U0z9u/WBYah/aOszf23sSN/9T+4/2P8AtpXrX7R0EyfGbxF5u/CRWf3/APr2SuniBcmFhUpe7znPkb5sZOnU9/kPMaKKK/OdT9E1Fjke3fej+XIn/LSvFvhP8GfDH7Qn7aniLw3r/wBqt7SS41Se6h0658l2eFC6Mj7H5L53p2r2mON5PuJ5lef/ALGsbL/wUi1c7Wxv1v8Ag/6YyV91wxL2dWvL+4fGcTR/cwX94+g9T/4Jv/s5aXfNbX/iHWrO5j+/BdeKLVJE/wCAPBUB/wCCef7Mv/Q26p/4V1l/8brV/aK0uO8+MXiF0XN0kVtvjdPv/wCip86V5X9nT+4n/fFKtxDiadScOxhh+H6GIpwqX3/rucP+1h4R8GfsZ6X4XT4TajHrMPio3q6suqX8Gp7Ps3keQUMYTZ/r5q+3f2rsN8NfCkneTUYJP/JGQ1+Y/wC2jHHHY+C9iY+e+/8AaFfp3+1XaTn4ZeEP3L4+3wf+kMlejjJxxOVRxvJ78ziw8JYbNFg3P3IHyzRRRX5rqfo4V81ftpf8efgr/fvv/aFfStfOH7asD/ZPBHyP1vf/AGhX0nDaf9p0X/i/9JPns/8A+RbV/r7R9s33/BNH9n/TtNtLvVNR8R6dbzJHsku/EcEKeY6b/L+eCqP/AA7u/Zk/6GrUv/Css/8A43XtX7VVuP8AhVfhd3X5P7QtT/5IyV8r/Z0/uJ/3xXu47PsTha3sz5vLsioYuh7e9jf+IP7HvwJ+DXw88T+PPCHiW6vPE/hvT5tV02C68Q2t0jzp9zfGkfz16P8A8E7fH+o/Er9mnxZrGqQ26XS69qFoPssflpsFjav/AO1K+cfihbp/wq/xl8if8gmb+D3r2z/glDaPefsleLxGv/M06p/6brGvQweK/tPBVMVOHvwOTG4V5ZXhRjP3JHn2h2b3lrAifu40gjkeST+CpLjyPPk8nf5f/TSpPM8vTrS1hR44ESP/AFn+sf8A6aVWr8ylo9T9OWwVzHxS/wCSV+M/+wTN/Ounrm/ilHJ/wqbxo+zCf2TN/OurBp/WKf8Ajic2Ma+rz/wGf+wb+xd8Ov2h/gjq/irxc+uRana+IbrTUOm38drH5CWtrJ8++N/+e717Yn/BO/8AZi2/J4r1L8PFln/8bqX/AIJVxt/wyT4xG3n/AISnUx/5T7GvONPt0/s20+RP9Qn8FfomdZxWwNbkpn51k+U0sdSc5M9Ib/gnn+zNsc/8JXqWf+xwsv8A43XA/su+NpdD/bmvPhfpH2Ofwl4eXWNL0+fYHneCCGTy98w+/wD6um/Z02fcT/viuB/Y1Xd/wUc1pFT+PXvkT/rjLWWV5j/a7qQrw+A3zDLv7JhzUp/GfRn7Sn/JaNa/69rL/wBEpXmdep/tKW80fxo1rej/APHtZf8AomvLK/P8YrYqZ99l7vhKf+AKKKWON5PuJ5lcaTudL0R4P8IPgxoHx4/bg8R+EvFIvP7KuLrVLhv7OufJffGkjp8+x/T0r661T/gm/wDs5aXfva3/AIh1qzuU+/DdeJ7WGRP+AvBXgH7HCv8A8PH9W+Rvva5/B/0wkr6H/aWt8/GrWtyZ/wBHs/8A0SlfquY5jWwGGpTh/LE/MMHl9LH42tTkUT/wTz/Zl/6G7VP/AAr7L/43Xhf7VHhnwf8AsV2PhT/hUeoxa4nid7r+1Bq1/Dqfl/ZvI8gp5YTZ/r5q7X7On9xP++K+bv2zo447HwXsXH7y9/8AaFedlmb1MzxcMLiYe5O/4K56OMyiOW4aeKo1Pfge8r/wRx1lmyPitpI/7hE3/wAcrzXxJ8edC+DXiHUPh7e6Pe6ld+E55PD099b3awx3DWpMEkiI4Ozf5dfoV+1VrmqaH4b8MvpWqXulvNfzI8ljdPBI/wC4P9yuL0Xxx8D20qzOvfD/AE3VfEEltGdR1C88KWd1JdXWz987zON7u7/PvfrXRiMdhsXN4fMV8Jy4bC4rCQVfBfbPnfwX4kg8ceEdK1+ztprW0v0kdIXfe6bJJE/g/wByjxxHjwL4t3phP7Fvf/SR63fip+xx8TvjX46vvGvwi8T6J4N+HuqR276TorahcaX9lSOCOFz9mhg2Jl0d/k/v14Qfhn8R/gf+1l8Kvh/478Vr4hh1XV9InuLW01Ge7sprWa/EbwSJIEL8o4dNmK5qfD8JVvb4etpvb9D0Z8QONL2FaHvn0b/wRvneLwL8TNjuD/a2n/db/phPXOW8aR+ZsRPvyV9A/tGaTafCvVfDlp4ItLfwTb3i3U91b+G4006O6dHjRJJPI2b/AC0/v14JXicQY2GJxNkd3D+DeGoOq/thXFfG3/kjnjX/ALB//txHXa1xXxs/5I541/7B/wD7cR14WB/3qh/ij/6Ue9jf91rf4JHn/wCzF/wT5vf2lvhgfGdp44s9AT+0ZtO+xz6dLP8A6tIzv3o+P467L4j/AAzT/gnnZ2Gma3qf/CbjxU73MD6dF9i+y/Zfk+ffv3h/P/8AHK+gf+CcMklv+w74onido7iPUdadJI32Oj/ZbetD4b+OvBsmm3J+KOlN45uvMT+z5tb0xNYktU2DzEj88fJvfZ9yv0/MsfQ9u8Hi/gZ+aZbhK/sfrOF+OB86fCz4xad8XLrVIdM0u7sv7PhSZ5JplfdvcJ/APeu+2P8A3HruPjL8J7P9pLRrLSfgNZ6L8O9Z0q6N3ql01kuhfarWSPZGm+1jff8APv8Av14T8b/2P/jB8J/hZF4pk8bpJc2csFrdWmmeIbyeS5kc7PMxJHHsrwXkOHxklUwdTkh2PoFn1fCw5MZD3zO+A7Ov/BTzwoAW3C7b7n/YLevtD413UZ/aE097z776FDHHJP8A89PMnrG/ZR8O6P8A8Mg+FPHE+k6f/wAJjLpN2z+JJLVBqXm/bpIP+PrZ52dnyff9q8t8Vtq663ONavbm/vv+Wd3dzu77P4PLd62ziv7CjRoL7H/tpz5RQeLr1sV/MbWoXl1p9jruiXlrdXF/e3XmQSRp5kb/AOr/APjdQap4XtfD/hzz9Snf+1pv9Rbx/wAFR2fxE1uzg8h/JuP+mlxH+8rC1DUJ9UupLq8n8yd6+QlXotn10aM0j5Y+Hf7Otx+1B+1F478JW2tQ+HZIrjU9R+1TWrzriO4xs2J/v/pXvGu/8E+779lbRb34rah46sfEFn4ZQXcmnWenyQPch5Eh2K7/ACL/AK73qD/gn3j/AIb2+In/AF467/6VJX058S/FkOl/tBapbeJZrzVfBSSQx6hoMn+lWk6PYp+7ktX+R/n2PX6dj8x+p0oQn8EuVH5xg8G8VipypfHD3j4l0L9qzw/4h1rTtMh8OalHcXlylqjSXsOxN77P+ede4SW7xySJsf5P9ivaPEGtfBbxZoOpaL4c+H+j6b4k1a1ex0vUG8LWVqbW9mj2QSecg3pseRH3x/cr5oj/AOCb/wC0rcTIkfxR0Le77P8AkZr7P/oivm55PgMx9/Bz9mfQU85xmB/3yHOebftix/8AFudB5x/xM3/9EivvLxErSfsH/CpX3SRDw/4aGyT7n/HrFXyr/wAEw4Y/i1458eWXjmC38cWVnpEM1tb+JI01FIH+1Im9EmD7Dsr1X4h6leQ+LPEOirf3iaLZalPb2ul+c/2W1jhndII44P8AVpGiAbP7mK3zBrKcuWBl7/mc+D/4V8w+tw9w5a3k+xyRvD+72fc8ur8kaapHJPCnlzp9+3j/AI/+mkdZ9LHI9vJG6P5cif8ALSvz0/QNj5l/bW/5kn/rjdf+hpX0BJ/wRv1mKZlPxU0vKH/oDTf/ABdeJftpwjV/+EQmt0UTw2t1JNDH02eYnzpX6c/teatqeieG9Fm0vU73TJ31d43exungkdPIk/uV+n4XGzwOUUJw/vf+lH5ljsPHG5tOD393/wBJPzn8TfHzQvg34i1H4fXmjX2pXfhOeTw9PfQXSpHcPakwSSIh+5v8v3r0/wAD+JYfHnhHTtfs7aW1tL9JHSGaTzHTZJJH/wC06+htF8dfA9tKspNe8A6bquvyQxnUdQvPCdndyXV1s/fu80g3u7vv+d68S+Kn7HfxM+N/jy+8a/CDxNo3g34eaokP9maLJqE+l/ZikCQzf6LDH5ce+ZJn+Trvz3rzKmDy3Ml/s8/Z1N7no08yx+A/3mHPAzPFUf8AxR/iTen7v+yL7/Wf9cJK7D/gjnK48FfEsLI6H+1tP5V+3kT180XXw1+JXwH/AGp/hl4F8c+K11xdV1TSLuaC01Ge6tJrWa8CGOSOQIX+44ZCn86+8v2jtJsvhRrHh+08EWsHgi2vUup7q38NQpp0d06PGiSSeRs3+Wn9+u2NFZDg5qc+fnOWvXee4qEIe4fP1vGkfmOiJ9+pqKs2dm95JsT93Gn7x5JP4K/NG23c/RUuVWOJ+Ltm958H/GsaMqRpp/mO7/cT9/HXE/s6/sC3P7TPwrTxLpXjWx8N6empTWIs7vTHmnlaKOM+Y7o/fzPufwV3Xx4vEHwW8Y2tsAkEdl0P35P9Ij+evcP+Casjx/sV6/KjvHKmp606SJ99H+ywV+kcP1p4fLq1SH85+d8QR9tjoQn/ACnzt8R/hkn/AATzt7DTNc1P/hOB4q33MD6dF9h+y/Zcp8+/fvD+f/45Tvhb8adL+LeoajZ6bpV3ZPYWqXTyTTK+/DhP4B/00r6N+Gfj3wbJpNyfihpTeOboSJ/Z82t6YmryWybB5iJ5w+Te+z7ntVL40fDDTv2lPC9poXwG0/Q/h34g026jvdUvm05NE8618uRETfaxu7/P/BWE4Zdm/vPStPqdNOrmGVP2b9+jA4HY/wDcevLfgc0kf/BUDwkCShFwmR/3CjWJ8eP2Vfjv+zv4Dn8YeIviPa3unw3MFs0Ok69eyTb5t+zh40/uetfWf7O3h3Srz9gzQfiDLplk/j5dP1KdfFbWsf8AaqOmozwo/wBqx53+p/d/f+5XXg8rjk8KmKlU5/c5Tkx2aRzZQwsIcnvGv+1gRc/Fq1eX95J/Ytr+8f8A66T15BVzU9TvdXuvO1HULm/n2eX593O88nl/88971Tr86r1/rFecz77CU/q9CFMK+XfAf7Ptx+0p+1b4z8F2utw+H5WvNTvvtU1q86fupnOzYn1r6irz/wDYR/5SHeLf+uWu/wDoT19pwvN03XqL+VHyXE/v0oL+8bWuf8E+b/8AZX0W++K9/wCO7LxBZ+GUF3Lptnp8kD3IeRIdiu/yL/rvevOdD/as8P65rem6ZD4c1KO4vLlLVHe9h2JvfZ/zzr7c+JXiyLS/2gtUtvE015qvgqOSGPUNBk/0qznR7FP3clq/yP8AP5b0zxBrXwW8XaHqGi+HPh9o+m+JNWtpLHStQbwpZWv2W9nj2QSGZBvTZI6PvSuytiMvzKf+2r3/AIThoU8wy6CeG+B+8eLyW7xySJsf5P8AYrwf9sWM/wDCutAyuD/ab/8AokV6Yv8AwTf/AGlpJET/AIWfoGXbZ/yM19/8Yp3/AATN0+L4nePviHpfjuG38a2+n6ZC9vb+Io01GGFvtQR3RJg4U/hRgMijgq0cWq3PyF4zPY4qjLC8nxn1Rrcksn7Bvwzjd3kiHh3wuPLP3P8AU29fPUcaRx7ETy6674hatfw+KvEuipf3keiWepT2lrpSTP8AZYIIZ3SCNIP9WiJgbP7mK5Ovkc2xX1uu5n1OUYX6rheXuFfMv7a3/Mlf9cbr/wBDSvpqvmb9tb/mSv8Arjdf+hpXXw3/AMjOj8//AEk5uIP+RfP+vtH0Av8AwRx1hcEfFbSuP+oLP/8AF15n4o+PmhfBrxBqPw/vNHvtTu/Ck8nh6e+t7pYY7h7UmCSREwdm/wAv3r9E/wBrLXNU0Pwr4cl0rVL3TJ5tQCPJYXTwSOn2U/3K4TRvHHwQbSbKTXvh9puq+IJLaM6jqF54Us7qS6utn793mcb3d3/jevqcTjsLipvD5ivgPlMHhcXhaKr4P7Z85+DfFEPjrwlp2vWdtNa2l+kjpDNJ5jp5ckif+0627/xHqPhzwt4gms5/LjTS7p/Lk/1f+okpfip+x38TPjf48vvGnwg8S6N4N+HmqJD/AGZosmoT6X9mKQpDN/osMflpvmSZ/k678968J1H4X/Er4CftUfC/wL438WLrsWsX+lXUlvZ6lcXVnLaTX3kmGRJEQvzG4dNnNclHh6E6v1nD1tN7foejPPuWn7CtR98+vP8AglD8SJvEvgv4gNJZx20seraerfZmxv3wSDP/AI5Tv+FoaXHJPssr3zN8n8Cf89P+uldh+0hpVp8J9Y0C08C2kHgi3vEup7q38Lxpp0d06PGiSSeRs3+Wh/jrwSuDPM2nKty0zpyLLYRpurU+2dxqHxUnkj2Wdklv/wBNLh/Mrx74/apdap8I/G015PJPJ/Z//LT/AK7x11NcX8bP+SM+Nf8AsHj/ANHx18/ha86mKo8/80f/AEo+hxVCnTwtbk/kked/st/8E/7z9p74YXHjG08cWfh1INVm0s2tzp7zsTHHBJ5gZHH/AD3H5V2vxG+GUf8AwTztrDTdd1L/AITf/hK99zA+nRfYfsv2X5Pn3794fz//AByvfP8AgmKzxfsY+MJI3dJE13V3SSP76P8A2daVq/DPx14Mk0m4PxR0pvHF6JE/s+bW9MTWJLVNn7xI3m+5vfZ9yv0jMsfR9s8Liv4bPzjLcJX9j9ZwvxwPnH4WfGrS/i3qGo2em6Vd2T2Fql08k0yvv/ebP4B/00r0HY/9x69A+NHwv039pTw3aaF8B9P0X4d+INNuo73VL5tOTRPOtXjkSNN9rG7v8/8ABXyx8dP2S/jz8A/hxfeMvEPxEsr3SrOaCKSDS9evJJ98z7E+R40//VXg/wBgYfGy9phKnJHsfQ08+rYWHJjIe+a/wNkMf/BUDwiD8jLcJ+H/ABKjX1l+1iwuvipYSXH7x/7Dh/eSf9d56w/2ddB0m6/YL0L4gzaXZP47Fhfzp4ra0j/tVHTUZ4Uf7Vjzv9X+7+/9yuE1LVtQ1abztR1C81OdE8tJ7ud532f88/nqM9r+zowwK+xH/wBJFkdB4nETxz+1Ip0UUV8Irn3LufMHg34A3H7Sv7VvjLwfa63D4fkkvNUvftk1q8yHyZCxTYn+eK971L/gnfqn7NOk33xRvfHdhr9n4Vi/tSTT7XTpoJLlEO3Ysj/Kv3/esv8AYbwP+ChHiv18vXv5mvqT4neLBpf7QWoW3iC5vNQ8FxyQx6hoMn+lWk8D2Kb45LV/kf56/YcdmH1OjShU+CcY/wDBPyfC4N4vFVpUn78PePijQv2rPD/iHWtO0yHw5qUdxeXKWqNJew7E3vs/5517hJbvHJImx/kr2rXNa+CnirRdQ0Xw58PtH0zxJqVtJY6XeN4UsrXyb2ePZBJ5yDemx3R96dK+ctN/4Jw/tFx3a/afiXodwh/dpbnxNqCb27f8u/3a+bqZRgMwXPg5+z/U+gpZxjMD7mMhznnn7WtvFZ/DvQL29h3R/wBrv5NrjZ5v7kc/7lfcevXEtx+wZ8MfN3Ff+Ef8NfJ/B/qIq+Xv+Ca1v/wtj4hfEWy8exW/je20/TIXtoPESJqUML/aQm9Em37PlzXpvxK1S9tfFviXQIb66g0DT9Te0tdHSZ47W2ggf5ESD/VoifwV1Zg1lOBWC+NnNhP+FbMPrcDj440jj2JTqKK/ONT9A1Pmj9tT/j38F/W9/wDaNe+t/wAEctXXB/4WtpY5/wCgLN/8XXgX7af/AB7+Cvpef+0a/Uj9rLVtQ0Xwn4cm0vULzTJn1HY72N08Ejp9lP8Acr9TwWNngcnoTh/e/wDSj81x2Fjjc3nTe/u/+kn50eJPj1oPwY8R6n8PbzSb7VLvwlcSaBPfQXSwR3D2shgkkROdm/y/evTfBHiiHx34S0/X7O2ltbS/SR0hmk8x08uSSP8A9kr6J0Lx18Em0fTz4j8A6brPiQ20H9qaneeFLO7nurry/wB5I80g3u7v/G9eJfFT9jv4m/G/x5feNfhB4m0bwb8PNUSH+zNFk1CfS/sxSBIZv9FhjMab5kmf5Ou/PevOqYPLc0X+zz5Knc7qeY4/AaYmHPAzPFUf/FH+JN6fJ/ZF9/rP+uEldJ/wRqkf+w/iqqPtzf6RuKv22XdfO+pfDP4k/AL9qn4XeBvG/i5dch1rUNKu3hsdSnurSa1mvvJMciSIhfmNwybOa+8f2jNFsfhTqWg2/gWxh8EW979te6j8MQJp32p0eNI/M8jZv2eY+zf/AH6640FkOEmpPn5zjxFf+3sTCnD3OU8A8tI7qd0T9558n/oynUUV+Zttu5+lJcqscf8AGX/kkPjX/sH/APtSOvNf2W/+Cf8AeftPfDC68Y2vjez8Orb6pNpRtLrT3nB8uOB/MDo//TcflXpPxl/5JB42/wCwf/7Ujr2n/gmOzx/sZ+MJI3dJE1zV3SSP76P/AGdaV+icPV54fLq1SH85+ecQ0o1sbRhP+U8G+IfwqX/gnjZ6faa7qh8cJ4xaR4H02L7F9l+y/Id+/fvD/av/ABynfC341aX8Wr3UbbTdLu7GSwtUunkmmV9/7zZ/AB/z0r6J+Gfj7whJY3J+KOnN44l/cnTG1rTE1j7Mmz9/5fnfc8z5Puf3Kq/Gj4X6d+0r4ZtdB+A+n6H8O/EOm3Md7ql+2nJovnWvlyIieZaxu7/P8+ys5/2dm65pe5Wn1Nacswyp+zfv0YHA7Hz9x68k+D+//h5t4QAHI1K14/7h1b/xt/Y7+L3wp+FbeKZPG4mudNeGC9tLDxDeTyXUkj7AYw8aAduK+nP2P/DulT/sd+GPHN9pOnyeNpNP1J38STWqDVfOTUp4Y/8ASdnnfc+T7/tXdgMsWUqpipT5/d5TizDNI5rCGGhDk946T44XMZ+PWkPfbpHfw/BHHJP/AM9PPnryrULy60/Tdd0S8tbq4v7268yCSNPMjf8A1f8A8brI8VvrTa5J/bV7eX92n3J7ud3+TzP+Wbv/AAVcs/iJrdnB5D+Tcf8ATS4j/eV8ZUxUKlSfOfX4fCTpU4KBJqnhe18P+HPP1Kd/7Wm/1FvH/BXxl4M+Adx+0j+1h4x8HW2tw6A8l7ql79rmtXmT9zI7lNiV9WahqE+qXUl1eT+ZO9edfsNc/wDBQzxZ/wBc9e/ma+k4aqfvK9Sn/Kj5/iGH7mHP/MbOpf8ABO7Uv2aNLvvile+O7DXrLwrD/aMunWunTQSXOwhdiyP8i8uPWvN9D/at8P6/rmm6XD4c1KO4vLlLWN3vYdib32f886+1/id4sGl/tA6hbeILm81DwXHPDHqGgyn7VZzwPYpvjktX+R/np+u618FPFWj6hovhz4faPpniPU7aSx0u9bwpZWpgvZ49kEnnIN6bJHR96dK3rYrL8xnbGr3/AITloU8wy6mnhvgfvHislu8ckibH+T/Yrwf9sWM/8K60DK4P9pv/AOiRXpi/8E3/ANpaSRE/4WfoGXbZ/wAjNff/ABinf8EyrKL4nePPiFp/juG38a22n6XC1pB4hjTUYYX+1BN6JMHCU8vyKOCrRxarc/IXjM+jiqM8LyfGfUmuSSyfsE/DKN98kQ8OeGh5Z+5/qIq8Bt/9DkjeH93s+55ddZ8SNUvbXxh4k0CG+uodAsNSe1tdHWZ47G2ghcbESAfu40T+CuSr5HNsV9ar859TlGF+q4Xl7mhJGmqRyTwp5c6fft4/4/8AppHXyx+2p/x6+DPref8AtCvpiOR7eSN0fy5E/wCWlfPn7adqurWvhKa3jWO4hS9kmhHR0/cfOld/Dj5szpP1/wDSTmzz/kX1v6+0e1L/AMEcdXXp8VtK+7n/AJAs/wD8XXmPib48aD8FfEWo/D680m+1S68J3EmgT30F0scdw9qTBJIiH7m/y/ev0b/av1jUdC8H6FLpuo3ulzPqcaO9hdPBI6fZZPk+SvOtD8dfBJtG0+TxF4B03WfEhtoP7U1O88J2d3PdXXl/vJHmcb3d3/jevqMTjsLip/V8x2Pk8HhcVhaft8F9s+dfA/iiHx54R07X7OzmtbS837IZpPMdNjun/slaOtx/8U7rpdP3f9nXX/oiSrfxN/Y/+JPxx8eX/jX4QeJdF8GfDzUkh/svRpL+bSvIMMKQzf6LBHsTfNHM/wAn9/Pevn/xZ8N/iZ+z7+0d8PfBXjfxcusvqtzp14yafqk9zayWs12YSjiQJ/zzf5dvQ150OHaVep7bDV/c7Hrf6wOFP2daHvn0f/wRtneLQfioEk2Zv9I3FX6jZd1mfZ0jurt0T9558n/oyvoP9pLRbD4T6p4ft/AthD4It71717uLwzBHp32p0eNI/M8jZ5mze+zf/fr5+rzOIMbDFYn2a+wdnD+D9lQdZ/bCua+Klm958I/GqIyqiaY8ju33E/eR111nZveSbE/dxp+8eST+CuU+OF4n/CnfGNrbAJAmn9D9+T95H89eFgf97p/44nvY3+BU/wAB5z+zb+wdcftQfC+58SaR4zsPDGn2WrTab9nudOeaeZkigfzndH/6b/c/g9811HxC+Esf/BPOzsbbXNU/4TaPxk8kkLadF9iNr9i+Q7xMHDh/tf8A45X0B/wS7/d/sg+KJEfy3TxDqzpIn8H+g2lWvhf4+8IS6bcH4q6c3jiUeSdPbW9MTWPsqeX+/wDL877nmfJ9z+5X6ZmWOpxrfVMV/DmfmuW4SvyPFYX44Hzj8LfjdpfxX1S+0/TdLu7J7K1+1SSTTK+5PMjjx8gH/PSvRdj/ANx69E+Mvw40X9pLwnD4c+BGmaL8O/FFjeR6hqGptpyaL5tl5cieR51rG7v87o+z/wCtXyn8dP2Svjz+z98OL/xl4g+Itle6VZywRSQaTr15JPvmfYnyPGn/AOqvBeQYbMJKphKnJHse+s+r4WHJjIe+aXwqYp/wU08Dg/IVv7Lj/uHJX1t+1g32r4pWD3O+R/7Cg/eT/wDXeesP9m7QdI1D9g/QvHt1pVjN47TT9SmTxQbSN9UR01C4hjkW62ecHRBs+/8AcrhtT1fUNWn87UdQvNTnRNiXF3O88mz+589Rntb2dGGBX2I/+kiyWg8RXnjX9op0UUV8Irn3LvY+XvB3wCn/AGlf2rvGXg+21uHQHkvdUvftU1q8yfuZHbZsT/PFe/6n/wAE79T/AGZ9Lvvije+O7DXrPwrF/aMmn2unzQSXOw7NiyP8q/f96yf2GAP+HhHir/d17+bV9T/FbxYNM/aBvbbxBc3moeC0Nt/aGgyf6VaXUD2qb45LV/kf56/YcfmH1OjShU+CcYn5RhcI8ViqkqXxw94+IdJ/aw8P6tqVpYQeHNSS4up1gRnvYdnzvs/5517pJbvbzyJsf5H8uvb9U8RfA/XtNu9L0X4eaPYazqET2tjcN4SsofJunTZA+9Bvj2O6Pvr5hX/gm/8AtKyTIq/FHQtzvs58TX2f/RFfOTyjAZgr4Ofsz36ecYzA/wC+Q5zzv9r5CPhfpOV2n+1uP+/L197ag0kn7Anw5R3eSIeGvDWIz9z7lvXyZ/wTb0tviB8ZvH+h+Po7fxrBpuiSPDZ+IIxqVvDOl7GnmRpMHHd/nH9+vWPiLqd7aeMPEnh+31G6h0Cy1Ka0tdHWZ47GGBH+REg/1caJ/BW+OkspwCwMvf8AM58HfN8w+uQ9zkOQjjSP7ieXTqKK/Oj9A1sfNX7av/Hv4K+l5/7Rr3eH/gj3q1zDHIPirpQ3qj86NNn503j+OvCP20P9X4N/3b3/ANo1+oX7TWrahofw38JzadqF5pkz30KO9jcvBI6fYpPk+Sv1PBYyeByehOH97/0o/McdhFjM3nRf93/0k/OzxL8etB+CviTUvh7eaTfapdeE7iTQJ76C6WOO4e2JgkkRD9zf5fvXpngjxRD488I6dr9nZy2tpeb9kM0nmOmx3T/2SvorQfHXwTbRtPk8R+AbDWfEhtoP7U1O88J2d3PdXXl/vJHmkG95Hf8AjevD/ih+x/8AEr43+PNQ8a/B/wATaL4L+HmpJD/ZmjPqE+leR5MKQzf6LBHsTfNHM/yf389686pg8uzL/d58lTuelTzHH5fpiYc8Cz/wkGo+G9D1p7ad440066keORP3f+okrt/+CTvxJm8TeH/iRHJZx280F7pCMbZsbw6XAz/5Dr5A8WfDD4mfs7/tGfDnwZ438XrrbatdadeSLp+pz3dq9s980PlyCRU3/wCpf5MV+gH7SGi2Hwl1Pw9B4FsbfwRBevevdxeGYI9O+1bHjjj8zyNm/Z5j7N/9+u2lSlkOFnzz5+c4K9WOe4mFOnDkOMvPihpcepX+yyvd/wBqn/gT/npJ/wBNKydQ+Kk8keyzskt/+mlw/mVxP+/Ta+AqY6s2ffQwNFKxzXx01O61b4T+NZryaS4kOmfx/wAH7yOvJv2VP2BL39qT4aX3i208bWfhxbXV5tKNrdWEk+8xwwSeZvRx/wA9wMe1enfGT/kkPjX/ALBg/wDRkde0f8ErZGj/AGQ/GTxt5UieJdUdZF/g/wCJdY191w/XnTy6tVh/OfD8QUYTxtGn/dPBviJ8J0/4J42Onwa7qn/CcR+MZJHgk02L7EbX7F8jhw+/eH+1f+OU34WfG7S/ivqmo6fpul3lk9na/apJJplfcnmRx4+Qf9NK+jfhd468ISabcn4qac3jiX9z/Z7a3p6ax9mTZ+/Mfn/c3/J9z+5UPxm+G+jftKeE4fDnwI0zRfh34rsbyPUL/U205NF82y8uRPI861jd3+d0fZ/9aueX9nZv71TStPqdNOWYZU+R+/Rgee7H/uPXkvwrYx/8FNvBQzsUX9ln/wAFyVU+Nv7I/wAevgL8OdT8a+IPiNZXuk6e0CSw6br968375/LTajxp39+ma+qf2b/D+kaj+wbofj250mxm8dR6fqUyeKDaRvqqOmoXEMci3Wzzg6JhPv8A3K7MDlscnU8V7Tn93lOTHZpHNlDCwhye8dH+1wRe/ErSHuN8jjRYx5j/APXaevFqt6nq2oa1NHNqOoXmpzomxJL6d55ET+589VK/OcVU+sV5zPvcHh/q1CFMK+V/CPwJuP2jv2svFPgu21iHQHudQ1S6+1zQPMi+SZH+4lfVFecfsR/8pE9Z/wCu3iH/ANFyV9nwpPkdeov5UfKcT/wYL+8dJqf/AATv1H9mnTL74pXvjuw16z8KxHUZdOtdOmgkudh2bFkf5V++PWvM9L/ax8P6rqlpp8PhzUknup0gjd72HYm99n/POvt74reLBpf7QV7beILm8v8AwXH9m/tDQZD9qtLqB7FN6SWr/I/z1Nq3ib4H69p11peh/DzR7DWdQie1sbxvCVlCYbp02QPvQb02O6PvrsrYrLsxnbGr3/hOGjTzDAU4PCfBP3jw+S3e3nkTY/yP5deOftZW8dn8MNEvbyHen9r/ALmH/nt+4f8A8cr0LTv+Ccv7RkV0v2r4laJPl/LW3/4SXUF3t23f6P8AcrO/4Jz2kvxQ+N/j3RviF5PjW107RJJIrTXETUYYZ0vYI96LPvRPk3px/BWmAyKOCrfW1U5+QrGZ7HFUZ4Xk+M+t9XuJbr9gf4erKWKt4Z8NDZ/B9y3r5zjjSOPYieXXX/ETU72z8XeJPD9vqN1DoFnqU1ra6Os7pY20CP8AIiQD92iJ/AlclXx2bYr61X50fVZRhfquF5e4V82/trf8ePgr/fvf/aFfSVfNn7aX/Hj4J/66X3/tCuvhv/kZ0fn/AOknNnn/ACLK39faPc7X/gj3q9xaxzr8U9NAeKOQf8SWfPzpvH8dee+LvjbovwH8Q3/w41DTLvWbvwnPJor6jazrBHdPCTHv2H7n5mv0M/ac1jUdD+HfhOXTdRvdLme8gR3sLp4JHT7DJ8nyVwfh7x18FX0PT5PE/gWx1zxGbaP+09TvPC9ndz3V15fzyPNIN7yf7b19ZicdhcVUeHzBfAfJ4PC4vC0FisH9o+d/A/iiHx54R07X7OzmtbS837IZpPMdNjun/slaOtx48O67vT93/Zd1/wCiJKufFD9j/wCJPxu8fX/jb4ReJNF8HfDvU0hGl6LJfTaX9mMMCQzH7LDF5ce+eOZ/k679/evn/wAXfDH4l/s8ftG/DnwX438XLrT6tdadeuun6nPdWr2z3zQ+W4kVN/8AqX+TFedDh2lWq+2w9b3N7Hq/6wOFP2daHvn0F/wRlmkSH4wlJHX/AEjQt230/wBPo1CNI9Y1J9n7z7dP/wCj5K98/aO0TTfhPqGhxeBbC28Dx3st6btfDFvHpv2nyfJ2eZ5GzzNm+TZv/wCeleAf79eZxBjYYrE+zR18PYOdKk63842uY+KP/JK/GX/YKm/nXT1zPxQ/5JZ4y/7BU386+fwf+8U/8cT6LGf7vUPJ/wBlT9gO8/ai+Gl94ttPG1n4cFrq82lG1utPkn3mOGCTzN6OMf68DHtXffEL4RL/AME8bHT113U/+E4Txi8hhbTovsRtvsWA4ffv3h/tX/jle9f8Er/3f7IfjKRGMckfiXVHSRP4P+JdY1d+F/j/AMKSWNyfippreOW8uGTTP7bsE1j7L+7/AH/l+ePk8z5Puff2V+n5nmFONf6piv4cj8zyvCV+R4rC/HA+dPhZ8btL+K+qajp+m6Xd2T2dr9qkkmmV9yeZHHj5AP8AnpXoux/7j16L8Zvhxov7SXhSHw58CNM0X4d+K7G8j1C/1NtNTRfNsvLkTyPOtUd3+d0fZ/8AWrwH42fsb/GL4XfCeXxR/wAJvHNcaV5MN7a2HiK9mkuXeTy/3aPGmPn7ZrwJZDhsdJVcHU5Idj6BZ9XwsOTGQ985n4eK/wDw8o8DKF+ZdW0z8P8AQozX3N8criP/AIXlob3+6R30OOOOSf8A56edNXOfsc+HtKuv2P8Aw341v9I09/Gp07VHfxJcWqNqW+PUZ0T/AEnZ53yImz79eaeK31p9ck/tq9vdQu0+5cXc7v8AJ5n/ACz31tnFf2FGjh19j/205cpoPGYqrin9o19QvLrT9N13RLy1uri/vbrzIJI08yN/9X/8bqDVPC9r4f8ADnn6lO/9rTf6i3j/AIKjs/iJrdnB5D+Tcf8ATS4j/eVhahqE+qXUl1eT+ZO9fHzr0bn2EaFSx8seHvgXP+0X+114o8F22sQ6C9xfapdfa5oGmRfJ8x8bE+le/wB5/wAE4dU/Z30u/wDiddePbHX7XwnC+sSabBps0ElykHOxXc/Ia579jX/lI5q/+/r3/oiSvq34q+LG039oC+ttcu72/wDB6G2GoaHI7z2lzC9qm+OS1f5H3/3K/S8ZmH1OjShU+CcYn5xhsJLFYytKn8cNT4j0n9rHw/rWqWmnweG9TSS6mWBHe9h/jfZ/zzr3SS3e3nkTY/yP5de36p4i+B/iLTbvStE+Hmj2Gu6hE9rp1w3hKyhMN06bIH3oN6bHdH318wr/AME3/wBpWSZFX4o6Fud9nPia+z/6Ir5+eUYDMNcHP2Z79POMZgf98hznnn7XqlfhjpOV2n+1uD/2xevu1mZ/+CfPw6SR3ki/4R3Qcx/wf8s6+Uv+CbenN8QPjT470Px4tv4zt9N0SSSK311BqUEU6X0CeZGkwcDq/wA4/v1698Tb+90/xp4o8OWl9dWvh3T797S20eGZ47G2gQfIiQD93GiVtjmspy5YGXv+Zz4P/hXzD6zD3OU4+3/0OSN4f3ez7nl1fkjTVI5J4U8udPv28f8AH/00jrPpY5Ht5I3R/LkT/lpX58foB81/trf8eXgn/fvv/aFe5wf8EfdXuLaOcfFLTVDxRyAf2NNn503j+OvIP2z7VdY03wjNbhI7uH+0ZJoezp+4/eJX6VftN6tqGifDfwnNp2oXmmTvfQo72Ny8Ejp9hk+T5K/T8FjZ4HKKE4f3v/Sj8yx2Hjjc3nRe/u/+kn54+LvjdovwI8R6h8N9Q0y71m78JTyaK+pWs6wR3Twny9+znZ+teleBPFkPj/wlZa/Z2ctraXm/ZDNJ5jpsfZ/7JX0P4e8dfBVtD0+TxT4EsNc8Rm1j/tTU7zwtbXdxdXXl/vJHmkG93/23rxT4pfshfEb48eNrzxl8HPEWh+C/h/fqiafov22fSfJeFBDO/wBlgi2Jvmjkf8a86pg8tzRf7PPkqdz0aWOzDLXbEw54FbUI3/sfUvk/5cbr/wBESVb/AOCMcjCP4v7JHUm40IfL9b/NfN3xG+G/xR/Zz+Ofgrwf408ZvrEuqfYr7y9O1W5ubV4Hunh2Pv8ALz/qJPl/Wv0L/aM0HTvhVqOhx+BbC28ER3st6bpfDMEelm58l4RB5nkbN+ze+zf/AH666eHWQ4SpOc+fn5Tjr4h57iadOHucvMeC6hGkesak+z959un/APR8lRU7/wBnqezs3vJNifu40/ePJJ/BX5w3d3P0dLlVjlfiRZvefC/xqiMqJHpE0ju33ErzX9mv9hO4/ao+GV/4k0jxhY+GNO0/V5tKFvdaa889yyQQSee7o/8A03+5/Bz1zXpvxgvE/wCFSeMbW1Gy0TTJuP8Anp/t16z/AMEpZPL/AGS/F0iHbInijU3T/wAF1jX6Lw9WnQwFapD+c/PuIYe2xtGnP+U8E+InwhX/AIJ42GnLruqf8JxH4wkk8htOi+xG1+xYDh9+/eH+1f8AjlM+F/xx0z4sa1e6bpulXllNb2v2qSSaZX3p5kaY+Qf9NK+ivhd4+8JyWFyfippreOG2Q/2Y+t6emsfZf3f7/wAvz/ueZ8n3Pv7KZ8Yvhvof7SHg+Lwx8C9L0T4d+L7W8j1C81RtLTRfNso43R4POtY3d/nngfZ/sfSueX9nZv71XStPqbU5ZhlL5PjowPPdj5+49eO+Bysf/BSfwCD+7I1TS/8A0kjpnxs/ZH+PXwF+HOp+NfEHxGsr3SdPaBJYNN169eb98/lptSSNO/v0zX1Z+zF4d0fVf2EdM8fXmjafdeO4dO1WWLxRNZI+qI8d3PGkkdzs84SInyJh668Dlcco58V7Tn93lOXMM0jm0IYWEOT3joP2uCL34l6Q9xvkcaLGPMf/AK7T14tVzU9W1DWp0m1HULzU50TYkl9dPPIif3PnqnX51iqn1ivOZ95g8P8AVqEKYV8u+HfgTP8AtHfteeKvBdvrEOgvcX2qXX2uaBpkXyfMfGxK+oP4a4D9jP8A5SPax/1017/0RJX2HC03CrXmv5T5jif+DD/EdDff8E3tT/Z60u++J9149sNetfCUD6zJpkOmTQS3KQ87Fkc/J+teV2P7W3h6+vrWyTw5qSPNNHCZHvYf7/8A1zr7o+L/AIqOj/H6W11m7vb3wdstv7Q0KR3ntLmB4PnjktX+R9/9yprrxX8CNUtZrHTfhvo1tqNzH9ntbj/hD7KEpO/7uOTeg+T53SvQrYrL8fN/XV+8j7p5tChmGAgvqnwT948KuLd7eeSDY/yP5f3K8P8A2vYz/wAKrsNy7T/bMf8A6Jmr0Zf+CcH7S0kwhX4o6DvZ9nPiXUP/AIxWX/wTfs5PiF8evGnh/wAerD40tNM8PXEi2eu41G3S4S+tU8yNJg6b/v8Az/7dGX5F9TrRxft+fkNcZnqxVCWF5NZn1yJGb/gn38O43Z5Iv+Ed0HMZ+5/yyr59jjSOPYieXXZ/E3UL3TfGfijw7aX11D4csL97W10eCZ47G2gQfIiQD92iJXHV8hm2K+tYpzPp8mwv1XDcvcK+a/20v+PTwV/10vv/AGhX0pXzX+2l/wAengn/AK6X3/tCu3hv/kaUfn/6Sc2f/wDItq/19o+8v2uvj98Pbnwr4XI8VWgEWpTHP2W6P/LEjsnuPzr5j/4X38OT/wAzhZ/+At1/8Yr3vQ/2dfCPjj7RbfGrSbvRdKtdj6ZJqeoPpkbzyf6yNH/j+SOOtQ/sV/sgj/l4sz/3OD//ABdepRwuBx0PbY6c4TPGq4vE5fL2GCjzwPItD/be0/wzo9rpelfFRLTT7WPZDBFpbybP4/47WuHvNQ8T/tB/tZfCP4h2N4fFuiaZrGj2c2rYS38sQ3qSSJ5eEf5fNBzs/jr6W/4Yt/ZCbH7+zH/c4P8A/F13fgL4Ofs7/DKCxTw7rmn2iWd7/aMO/wAU+dsn/d/7f/TOOvSprD5f7+ErTn/j+H9Dz6lSrjvdxNHk/wAJzP7eHj7w34J17wO/iDWodLFza6j5HmQzPv8A39v/AHEevlkfHz4bkH/isbM/S0uv/jFbv/BWvxNoviq6+FEuk6zY6tHDb6osjWFyk4TM0GMlOnf8q7T4c/sL/CXTdW1gfErwtqHhfTdscdhPrOtPZJNNvk3oj5+f5AKyxWV5dUUK9eU7z/l5fsmuFzLGUqc6FGEfc/8AAjy7/hfnw46/8JjZf+At1/8AGK8S+I3xM8U/E7x5qng3wPrH9r+H9W2QW1rHCkazrsR3G90R/vo3XHSvv7/hiz9kMHH2iy/8LBv/AIuqnib9nP8AZn+G3hW+8ReGNXsbXXtGh87T7qfxUbhIpN+z7nmfcy9a4TDZXgeeth/fn9nn5dyK+Ox+P5KFeHJD+6Yn7HfiTSfgr+yL4r8M+NdRi0LWzJq119jbfP8AuXtY9kn7nen8D/lXisPx6+HEcaK3jC2UhMYNrdf/ABivWPB3g8/ESNJILObW/Bc11/Zuratpz+ZaJa/cuv36fc2QP9+u5/4Yt/ZEbl7mxz7eMXH/ALPXnxWHzSU55j7k/wC7/wDbHdJzylcmX+//AF/dPC/C/wC1p4W8BXF1PoPxDh0q4uUSOaRNOnfekf3PvwPW/wDtBfGXX/id8O9N8N+HvG0ni7xRNfQaodLECQRtDCJN/wC88tP78f8AHXqsn7F37IcnJurIn/scH/8Ai67TQ/2e/wBnGHVLW907V7U6xbWr2sf/ABVskheN/wDtpXo0MPhMJFTw1afP9nn+E4K2KxOLl/tVH3P8PvFfw3cf2H/wT1S+1uIaNcWegTrd27/P5P8Apr/P8m+vma4/ao+GWqaC9hq3im3uLyH/AI9LqO0uv/H/AN3X1B+1BrHgnwB+x78RvDejeIdNnkj0J7W1sTqiTz7HnQf77/6yvgz9lj9mHRPi14J0vX9a8J6xqNidda1vNQtZ3S3S1R4PM3v/AAbEeQ761xOFoYmn9axXN/L7hngsRXw83hsN/i9467/hfnw4/wChxsv/AAFuv/jFcD8Zv2itNsvDunv4B8WrLq32nFz9ntZMeTs/6bx/36+zv+GK/wBkE/8ALzp//hZSf/F0f8MU/sidp7H/AMK+T/45XJh8DlGFrqqpSm135bHTWzTNcRD2fJy/+BHzh+wFZ618P/j1e/EXx4E0/R9b0K6l/tKR0fzZroxuh8uEl038n7leh/tBfHr4dx/GPxM8vi23TebYjNrdf8+sP/TD2qS10/Rb7xtqngzwHcQ64NImntrXTtKuftsyWsEnlp/t/c2fPXoFj+yb8BPF2n2uqfFCJdO8dzx/8TG0vvEj2U8YT5I98Gfk/cohrnlWWaV3Tx6cIdOX8PwOhUv7KgsRgnzznuj5+s/2ifh7p97b3Vt41tbe6tp0ngk+xXXySJJ5kcn+or0G1/4KCJHdRyN8YP497j+yP/uSvSP+GK/2Qf8An4s//Cwf/wCLpB+xR+yCDkXVnn/scD/8XXRTwOV0v4daf/kpz1Mxx9X+Jhof+AnmP/BLb4W+KPhz8RviE3iPTBYR3GiwRx4mhkz/AKUh/gf2rM+LHxq8B6T8VvHNhfeKLezvrbxBqEE1vJa3R2Ok8gfomOtfaHgdPhF8PdUvb3QvFekwXF4iQzG48QJP8m/f/wAtJK/L+P4Z6f8AHD9tn4wab/Zd54mtG1vXb6CPSZD8/wDpr7JN6/wfOOfeuirQpZtCdbG/Y/kMaNSpltflwf2/5vkeg/8AC/vhv/0OVl/4C3X/AMYqrq37QXgD+xdS+w+MbX7d9in+y+Xa3X+v8uTy/vwf39le8+Bv2If2crbwrp0fja0XQ/FGyQ3ljqXiaS1uIfnfZvhz8n7vY9dCf2KP2Rf+fuw/8LB//jlefTyzJ4v2nPP/AMlO2ebZm1yKnH/yY/NlYfil+0FEmoL52vxaPJ5KTb4bfyGf5+Puf3P0r9Rv2vfjp8PtV8K6M9v4tswIdXed/wBxN9zyJP8AYryP4m+A/gr+z7JpVr4D13SNOtNVSaa9SXX/ALVvkhdEj+/9z78ldZoH7PvhvxddXEHxk06/8OaVZ+XNplxfai+l77r+PY+fnfZW2Mxn1issFyclDvFf1Ezw2HVCl9d5+av2/r3jwD/hfvw3/wChysv/AAFuv/jFdl4f/bZ0vwzpFrpWj/FNLDTbb5IYY9LeTZ8+/wDjtP77/pXsY/Yr/ZHvpJGQ2e/77pH4wc/+1Kh/4Yp/ZD/5+LL/AMLB/wD45U08tyqn8Faf3x/yCpmmY1tKlCJ8za1eeJf2hf2ovhV8Q7G/Pi3RdL1XR7ObVPkg2eTeiR08twj/AC+Zn7n8dfV/7eXxA8NeC/EXgqTX9ah0sXUOo+T5kE77/wB9H/cjeuj8CfBn9nX4Z2tpBoOtafaR2t9/aMEbeKvM2T/u/wDpp/0zSvlz/gq9rGleOdS+FZ0LWNO1SO1ttUa6ns7pJkhTzoPv7K9KnCnmNsFOf7tf+BnnyrVMDJYyjC0//JTFsPjj8O7+R408Y2UaIm93ktbry0T/AL8U+8/aK+Gnl/Zbbxjax2icf8et18/+3/qK9N8D/sV/Ciz1TVbP4geF9T8LeH0jjFpPrWszWcd1cFzvw+Ruk2Y+T612v/DFf7Ig/wCXqx/8LB//AI5XkU8ryiUfjn/5KexUzbMoz0hGf/gR8CfEr4m+Jvif461LwX4H1gaxoGrRww29tHEiLP8AuUeQB5kRx86OecdK+3f2L/E2k/BH9lXxF4Z8a6jDoes/adWuhbNvn/cvaR7ZN8O9P4HqfxZ+zV+zN8KfC+qeLfCWoaZaeINJi8+znk8UmdEk3on3PM+f79c54L8Kn4kW8dxZ202ueEJroWOp6rpj74EgPlpdfvk+5sgf7/8ABXbi8asPTp4XBU/3fV2OXDYT63z4rG1LTPJrX4+fDiG3hWTxja5CICPst1/8Yrf8LftZeE/A93dXOgfESDS7i4RIZpI9Pnfem/f/ABwPXun/AAxX+yL/AM/Wm/8AhZP/APHKP+GKP2RP+fqx/wDCwf8A+OVz08tyqm7wrT++P+RdTNsxqLlnQi1/28fOXx0+PGr/ALTXw1uvBOg+Mf8AhMtQe6hvf7O+yJagJCJN8m94I/74/jr6T+Etq3wx/wCCZ9tD4oH9kPpulX32rzD5nl79Xn2fc3/89E/OtXwd+y/+y14E1X+09F1Ozs7t4Xg3jxgX+R/+2lP/AGpNc8CaN+xr8QvDHhrxJpU9tDpQS3s01ZLm4P8Apsb/AN/L/feu6LpWWBjPmoz/APAzgrOTaxip8k4/+Anyp/wv34b/APQ5WP8A4C3X/wAYqP8A4X/8Nv8AocrL/wABbr/4xXJfsr/sw6H8WfBWmeINZ8Jaxf2La61pe6hazulvFbI1v5m9/wCDYjyHfX1//wAMW/sirGP9K04+w8ZP/wDF150soyqnOdOcp+5/hPTWcZlyQnCEJ83+I+MvjJ+0Rp9r4e09/AXi2KTVjc4uPs9rJjydg/57x/3812X7B2j694I/aIT4keOQtho2q6TeXT6rK6OZ3uo96Hy0y/z7/wC5X0z/AMMU/sh9RPZ5/wCxwP8A8crze3tdBvvHGqeC/Atxba62lyz2dnpmnXv22eO1tX2J/tv8iJXXPE0cuwfs8uhz9/6ic1GlPNa/PmD5P6/vC/tCfHz4dJ8aPE7zeL7ZN32XGbW66fZYP+mHtXn1n+0T8PbG+tLy28a2sF3azxzwP9iuvkkSTfHJ/qK+gNP/AGUPgP4wsbfU/ijFHp3jmeP/AImFpfeIXsZ4wnyQb4M/J+5RKsr+xf8AshsxBms/x8Xv/wDF1xxy/La/7+tOfP8A9unRLMsfQX1elRjOEPxPOLP/AIKAqmoW8s3xfyqOjuP7I/8AuSmf8Ew/hn4o8E/Ef4jXHiHTPsiXukQ+XIJ4ZN5+1b/4H+tenQ/sYfsieW8ay2AjY558Yv8A/HK9v8G2/wAJPhnfXV7oXifSrSS/hjgeS78QJPG6I/8AtvXoxdLCL2eEq88J/Fz/APtp59RyxavWo8k4fDy/+3HxT8VvjV4E0n4reOdPv/FFvZX9rr+oQT28lrdHy5EnkD9Ex1rkf+F/fDcf8zlZf+At1/8AGK85b4c2Hxq/bf8AixpS6Xc+JLSbXtevY49Kl++FupXR1dP4Dx+dfW3gb9iP9nK38KadH43tF0TxOEc3ljqXiZ7W4h+d9m+HPyfJsf8A4HXJiMnyunW5Kk5/z/ZOyhm+PdFOnCP/AJMeDav+0B4B/sXUvsPjG1+3fYp/svl2t1/r/Lk8v78H9/ZXzi1r8Tv2iYGvSG1620dvJEha2g8nfliP4N+dhr9KD+xB+yR5Pm+bZmP/AJ6f8Ji/+NcH8UPAfwW/Z/fSrXwBrelWFnqiTTXsc3iD7VveF0SP7/3PvyflXTReEyuhOWXrnqf3tWZuric2rQhjvch/dPUv2uvj98Pbjwb4dP8AwlduPL1P95/ot1/z6yD+57j86+Wh8e/hy3TxhZ/+Al1/8Yr3vQ/2e/Cvjf7XZfGrSrvRdBtSk2mvqV6+lxzXT/6zY+fn+QCtM/sV/sgj/lvZn/ucH/8Ai68ujhcDj4e3x0pwmdcsXicvl7DBR54Hkfh/9trTPC+kWuk6P8U00/Trb5YYY9LeTZ8+/wDjtP77/pXC+INS8SftF/tPfCX4hafqJ8WaJpeo6Xp0+rfJAEaC+850COEfhJlf7n/LSvpf/hiv9kJhxPZD6+MHH/s9dv4E+C/7Onwwt7SDw9rOn2kdre/2jAG8VGT9/wDu/wDpp/0zSvTorD4C88LVk/8AH8J58qlXHe7iaPJ/h+IwP29vH/hvwV4i8Ff8JBrUOli6g1EweZDO+/8AfR/3I3r5YHx8+HBHHjCzP0tLr/4xW5/wVu8VaJ4qvvhTJpGsWOrRwW2qLO1hcpPszNBgOU6d/wAq7D4a/sO/CjT9U1EfErwpqXhnTvs8As59Z1aaxjlnLneiPkb32Vjissy+ooV6853n/Ly/ZNcHmOMpU50KMI+5/wCBHmX/AAv34cf9DjZ/+At1/wDGK8T+JXxK8U/E7x3qng7wNrI1jw/qscENvaxoiJN+5jeQb5kRx86N1x/j99H9iv8AZD3cz2eP+xwf/wCOVmeLP2bP2ZvhT4W1TxX4R1DTrTxBpMXn2U8nik3CJJvRPub/AJ/v1thcNlmBc61D359Obl36BXxmPx/JRrw5If3TL/Yj17TPgf8Asr+L/DvjnUIvDmsyX+p3yWrl5z5L2MCJJ+538fu5P++K8Wtvj18Obe3hWXxhbBgiAj7Ldf8AxivU/BHhv/hZNrHNYW02u+DproabqepaXJvghhPl/at86fc2Qyb/APYrv/8Ahi39kRzl7rTs/wDY4uP/AGevPiqWaSnLMbwf93/7Y65OeUx5MvfP/X908M8K/tZeFfAd5dXOgfESHS7m5RIJnj06d96b9/8AHA9Zfxu/aE1j9pH4aan4G0Txq3jG/vJobk6cLRLXekLGRn3vBH9z/rpX0NJ+xZ+yIpG25sSf+xwf/wCLra8K/stfsueA9VTU9G1Kxs75UeHePGG/5H/7aV108PgMI/aYWtPmh0fw/M5J4zFYv3MTRjyz/u+8Z/wjhPwx/wCCaFtB4oB0h9P0q+Fz5h8zy9+rz7Pub/8Anon5180/8L++G46+MbL/AMBbr/4xX1b+1NrXgLRf2M/iD4Z8NeJNJuIIdKCW1pHqyXVwf9Ojf+/l/vvXw3+yf+zPonxb8G23iHWfCurapp0PiD7DeahazulvDbILd5C7p9zYjud9aYrBYXHUfr2Kcv8AtwywWLxOCm8Nhv8AyY7T/hfnw4/6HGy/8Bbr/wCMVwXxm/aI02y8N6c/gHxZFJq32pxc/Z7ST/U7B/z3j/v5r7Rf9iP9kHyxN51kIH+5J/wl8n/xdR/8MU/sidp7H/wr5P8A45WWHwOUYWuqqlKfrytG1bNM1xEPZ8nL/wCBHzJ+wjp+v+C/2hofiX48K2WiarpN5O+qTOjGeS5j+Q+WmX+c/wCxXqf7Qfx6+HX/AAuPxM83i63QObYjNrdf8+sP/TD2oWw0GTx3qvgjwHeW2vJpc81rZ6dpt59tmS1h+T/ffYleiaf+yV8A/FWk2etfFW2TTvG9zHm/tdR8QvZToEfZHvg8z5P3KJWU6qzSu6GPThThty/12NVS/sqKxGDfPOZ4Tpfx7+Heiy2Oqv40tUlgmS4sh9luvM3xvlJP9R9zfHXZWv8AwUCRb23kk+L2Y1mR3H9j9t/P/Lp6V6dd/sc/sk30xmuLqzkc/wDPTxm5/wDZ6r/8MU/sg/8APzZ/+Fh/9srWngMspfBWn/5KY1Mxx1b+JhoHm/8AwTB+GvirwP8AET4jXPiHTRYx3mkQ+XJ58Mm//St/8D+1ZPxf+NngPSfjD48s7/xRb2t3b65eQTW0lrdHy5EnkDjhMdfSvs7wWvwh+HOoXt74f8VaVb3F5DHBMbjxAk/yI+//AJaSV+X938ObH41ftzfFbS10+48S2c+ua3dxx6VJy4SeR0dHT8PzrplRp5vSnPG/Y/kOehVnlta2Gh8f8x6D/wAL++G//Q5WX/gLdf8AxiqurftBeAP7F1L7D4xtft32Kf7L5drdf6/y5PL+/B/f2V7z4G/Yh/ZytfCunR+N7RdD8UbHN5Y6l4mktbiH532b4c/J+72PXQn9ij9kX/n7sP8AwsH/APjledTyzJ4vn55/+SnoTzbM2uRU4/8Akx+a/kfEz9oqAzFW8QW+ivs3F7a3MPnf98bs7P0r9P8A9rz4/fDy48G+Hf8AirLfEepgyH7Ldf8APrIP7nuK8w+JvgH4Kfs8rpVv4C13R9Ng1nznvPN1/wC1b3g2eX9/7n+seuk0P9nnwp45e8sfjVpd3omhWpSbTX1K9fS45rp/9ZsfPz/JW2MxSxFZYLk5KHeP9cvxGeHw/sKP13n5q/b+vePBP+F+/Df/AKHKy/8AAW6/+MV2OgfttaX4Y0e10rSPipHp+m2+/wAmGPS3fZ+83/x2n+3+levf8MV/sh/897L/AMLB/wD45R/wxT+yH/z8WX/hYP8A/HKmnluVU/grT++P+QVM0zKrpUoR+4+aNc1DxF+0b+098JfiDYX/APwlmi6Tqel6fPq3yQCPyL7znQRuEc4SZX+5/wAtK+tP28viD4a8F6r4Gl1/V4dLiuo9R8kyQzvv2PB/cR62/AfwX/Z2+GEFpB4e1nT7SO2vv7Rg3eKvM/f/ALv/AKaf9M0r5a/4K4+KNF8UQ/CGTRNYsNXSBdXWc2Nyk/lkvaYDlOnQ/ka9ONOnmVsDOf7tf+BHC608FJYynC0//JTnP+F/fDf/AKHKy/8AAW6/+MUf8L++G/8A0OVl/wCAt1/8Yr0z4a/sN/Ciz1LUR8SvC2o+GNO+zQizuNa1Z7KOWbed8aOCN77K9GP7FX7Ii8farE/9zg//AMcrxI5Xk8o83PP/AMlPXqZtmUZcqhGf/gR8B/Ez4o+JviR441Pwd4G1gavoOrRwW1vbxxIiz/uUaQB5kRx86P1x0r7S/Yh17S/gZ+yz4v8ADvjm/h8O6xJfanfJbNvn/cPYwJ5n7nfx+7f/AL4rovEP7Nf7NHw/8M3viLw5q2n2mu6Lavc6dczeLXnSF+3yeZ9zL1x/g7wi/wARPLNpbS674Te6/s3VNR0t/MtEgfy/P/fJ9z9zJv3/APA69DFY5YenTwuCp/u+rOXDYb63KeKxs7TPI7X49/DiG3hWTxja7giAj7Ldf/GK3/Cv7WXhTwNdXV1oHxEh0u4uE8maSPT533p5m/8Ajgevc/8Ahi39kT/nvp//AIWT/wDxyl/4Yp/ZD/5+LL/wsX/+OVzwy3KqbvGtP74lVM3zKorToxseX/H7416/8RPhta+GfDvjiTxX4pvJ4dR/stbRII2hhcyN+88tPuEf3+1e++CbqTR/+Cer32sxf2LPY6HfNfQP8/k/6bP8/wAm+rOg/s7/ALN/9rWN7p2r2v8AbFnC9rH/AMVc7l43/dv/AMtKsftE654G+H/7IfxK8L6P4g012/4R+6htrH+00muDvf3fe/zu9evTdNr6pTn+6n/4EeVU6YnktWj/AOAnytP+1R8MdW0B7HVPFVtcXkP/AB6Xcdrdf+P/ALuuZ/4X98N/+hysv/AW6/8AjFcZ+yX+zRonxc8G23iDWfCuq6pZReIRY3uoW07pbw2yC3eQu6fc2o7nfX2I37FP7IoUf6Xpp/7nJ/8A45Xk18ryqM506k5+5/hPVhm2YQhGdOEXzf4j4z+Mv7RGm2vhvT38A+LYpNWNzi4+z2kn+p2D/nvH/frsv2EdP17wZ+0NB8SvHm2w0XVdJvbp9VmdHM73SfIfLTL/ADn/AGK+m2/Yt/ZD6iayz/2N7/8AxdebR6foEnjrVPA/gK8t9d/suaa2stN068+2zpaw/J/vvsSt54ijl+E9nltPnf8AXWJlRpVc1r8+YPkX9fzCftCfH34cx/GjxO0viy2XcbXGbW6/59YP+mFef2X7RPw9sb63vLbxrawXdtPHPA/2K6+SRJN8b/6ivoCx/ZP+BHjCxt9T+KUUem+NZ4/+JhZ33iR7G4jCfJBvhz8n7lEqwv7Fv7IhYgz2X/hYP/8AF1xxy/La8VXrTnz/APbp0SzHH0P9npQjOEPxPObP/goAqahBJN8XsokyO4/sj/b/AOvSmf8ABMP4Z+KPAvxE+I1x4k0wWEd1pEPlyCeGTf8A6Vv/AIH+tekf8MWfsiYwtzaY9/GP/wBsr3HwOvwg+HN9e3ugeKtJguLyFIZvP8QJP8iPv/jevRhUpYRezwlXnhP4uf8A9tPPqRli1z1qPJOHw8v/ALcfGHxg+NfgLSfjD48sr7xPb2t3b65ewTWstrdExyJPIHHCfyrj/wDhfvw3/wChxsv/AAFuv/jFeeXXw6svjT+3N8VtKXTrrxJZz69r19HHpcnLhZ5HjdHTrnj86+s/Af7EP7Olr4RsR46sl0XxRiT7ZYaj4nktbhPnfZvhz8n7vY9ceIyfK6dbkqTn/N9k7KGcY+VDmpwh/L9rmPCNQ+Pnw+/s69+z+MbX7WbWfyNlrdf6zy38v/lh/fr5t2/FD9oSEXkbSa7DosmxH3w2/kNN/wB8dfL/AEr9KB+xT+yH3uLH/wALF/8A45XEfE7wB8E/2e10m38Aa3pNjb6x5z3vneIPtW94PL8v7/3P9ZJXTReCyqjOWXrnqf3iHVxOa1oUcd7kP7p63+1t8dPh9qXgPR5IvFNmHg1Te6eTN/z6yf7FfJo+PXw5bp4ws/8AwEuv/jFe++HfgF4e8WXE9v8AGLSb/QfD9r5c9hdX18+lo97z9x8/O+wO9bP/AAxX+yJeyO6NZmT77xx+MJD/AOz15lHC4HHQ9vjpzhM6pYvE5bL2GCjzwPG/D/7bWmeE9Hh0rRviomn6bbb/ACYY9LeTZvfzH+/af33Neb+N7/xR+0h+0R8NfHmm33/CY6RpN1pulz6odlrhob3zHTy3CP8A8vCH7n/LSvqX/hiv9kNhxcWX4+MH/wDi67XwR8Ff2dfhvY29roOtWFpBDe/2jHu8VGT9/wDu/wDpp/0zSvUpRw2ATngquv8Af+E891KuO93E0eRf3fiMb9vbx94e8E6z4Jk8RatDpcV0NU8jzIZn3/PB/cjevl+w+OHw7vpHjXxjZxoib3eS1utiJ/34rof+CsWuaX4+b4SHRdY07VY7WLV5bmayuUnSFC9p9/ZXV/D/APYp+FdpqV7bePvCuqeGfDnkxi3m1rVprJLm53no+fmk2b/krDEZXl9RQr15zvP+Tl+ya4XMcZRpzoUYR9z7zza8/aI+Gnl/ZbbxjbR2iHH/AB63Xz/7f+orwf4kfFDxP8SPHmpeEPAWsjWPD+rxQW1vaxQoizfuY2dN8yI4/eI3XFffB/Yr/ZC3cz2f/hYP/wDHKz/E37M/7Mfwx8Mar4q8J6hp9p4g0m1e6sp38WeeiT/8s/k3/P8AfrXDYbK8Dz1qHvz6c3LuRVxmPxnJQqQ5If3TN/YV17TPgb+zL4o8PeOdRi8PaxJqupX6Wc2+ffC9jbosn7nfx+7k/KvELX49fDm3s4YpfGFsJQiA/wCi3X/xivVfAvh2T4k232jSbKbxF4Ve6/s3UdS0p98CR/u/PTz0+5+4k3/7Fd9/wxd+yI53Pc6dID93HjJx/wCz1wR9nmnPLMvcf93/AO2O6TllMeXL3z3/AK+yeF+Ff2svCvgO/nufD/xEh0u4uU8id49Onfenmb/44HrL+N37QmsftKfDTU/A2h+NW8Y395NDc/2cLFLXckLGRn8x4I/uf9dK+hpP2LP2RFI23NiT/wBjg/8A8XW14V/Za/Zd8D6omp6Nqdlb3qoybx4wJ+Rxh/8AlpXXTw+Awj9pha0+aHR/D8zknjMTirwxNGPLL+77xQ+C1m/w7/4Jsw2niQHTJdO0nVWuvM/eeX/xNJ/7m/P36+aP+F/fDdevjGy/8Bbr/wCMV9cftGa58P8Aw/8Asf8AxE8O+G/Eukzwx6HMlrax6slzcPvnR/7+X+/Xwb+yn+zJo/xg8Ewa/qnhTV9ZtF8QNY3V9aTusCQILeSQSbPufJI/z1risFhcdReMxTl/24ThMVicHN4bC/8Akx2//C/Phx/0ONn/AOAt1/8AGK4b4x/tFaXZ+FbV/Avi2KXWfto877PaSf6nY/8Az3j/AL2yvsn/AIYr/Y+Zm2XOnhP+xyf/AOLo/wCGKf2RO09j/wCFfJ/8crHD4HKMLUVVSlP15WXWzTNcRD2fJy/+BHzL+wfpmveC/wBoaD4l+PCtlomq6TeTNqkzoxnkuY/kPloS/wA5z/BXqn7RPx6+Hf8AwubxA8viq3hBWzx/ot0f+XVPRPpTY7HQJPHWq+CPAV5ba9/ZU81rZ6dpt79tn+yw/J/vvsSu/sf2UfgP4x0+21T4rQrp/jqaP/T7S98QvZToE+SDfBn5P3KJWNWss1ruhjoOEIbcv9djohSWWxWIwb56k+h8/wBv+0N8OrO7t7qHxrax3ELxzxyfZbr5JEk3x/8ALCvQrX/goEq3sckvxfzGsyO4Gj/7f/XpXo//AAxX+yD/AM/Fn/4WD/8AxdOj/Yt/ZBCuonsPm/5af8Jifk/8frWngMrpfw60/wDyU56mZY+r/Ew0P/ATyr/gmj8OfFPhL44fELV/EOntb22paLujnE0Mm8vewSfwP7GqXxg+NngLSPjB49s7/wAUW9rdW2uXsE9vLa3UnlyJP5bjhMdj09K+z/Ben/CH4Y6hNfaF4n0mzku7aO1eS88QJPG8KP8A9NJP9ivy2vvAFj8Zv26fido5sLjxHZXniLXruOLSpMmbZJcSRujp17fnXVKjTzelOeN+x/Ic1GvUy2vbB/b/AJj0n/hf3w3/AOhysv8AwFuv/jFM1D9oL4e/2dffZvGVr9q+yz+R/ot1/rPLk8v/AJYf39le5eAv2If2dYfCdgnjixXQ/FGJDeWOpeJpLW4hy77N8Ofk/d7Hro/+GHf2SfI877RY+X/z0/4TGT/GvOhlmTp8/PP/AMlPQnnGZxXIqcf/ACY/NXyfiZ+0VbmXDeIbfRTsyXtrcw+d/wB8bs7PfpX6eftVfHj4ef8ACtvDEbeKraNoNQhRz5Fy/wByykT+59K8y+JvgD4Kfs+rpVv4C17R9Nt9Y897zzdf+1b3g2eX98/J/rHrotD/AGfvDHjxp7T4yaVdaR4XhRJrGfU7x9MSa65/1b/x/uQ77K2xeMVessFycuH7x/rl3MsPh1Ro/XXPmr9v6948H/4X78N/+hysv/AW6/8AjFdj4f8A22NL8K6PDpWj/FNLDTbbeYYY9Ld9m9zI/wB+0/vua9e/4Yr/AGQ/+e9l/wCFg/8A8co/4Yp/ZD/5+LL/AMLB/wD45U08tyqn8Faf3x/yHUzTMa38ShF/I+XfG994l/aS/aC+GfjvS9Q/4S/TNLvNP0yfVsJahHjvnmZAj+W/3JlfOz+P2r6+/b2+IHhrwXrHgmXX9Xh0uK6GqeT5kMz7/ng/uRvWz4H+CP7OfwztYrbQda0+0jhvf7Rj3eKvM/f/ALv/AKaf9M0r5e/4K5eKdE8Vr8JJNE1iw1dIF1dZ2sLlJ/Ly9pgPs6dD+VenGnTzFrAzn+7X/gZ58q08FJY2ELT/APJTm/8Ahf3w3/6HKy/8Bbr/AOMUf8L++G//AEOVl/4C3X/xivRvhf8AsO/CyG/uZfiP4W1Lw5oz2cAtrrWtWeyjlun/AII3yN8mwO+yvTD+xV+yIv8Ay9WJ/wC5wf8A+OV4scryeUebnn/5KexUzbMoy5VCM/8AwI+BfiV8UPEvxJ8daj4P8B6yNX0DWIoLa3tY4URJ/wByjum+ZEcfOjdcV9pfsG65pfwJ/Zp8XeHfHmoQ+H9Xm1fUL2O1c+fugfToE3/ud/H7mT/vitTxN+zP+zJ8MfDOq+LPC+oadaa/o9q91ZTyeLPPRJ8fu/k8z5/v1yvgfw4/xKtxPo9tN4h8LSXX9nahqOjvvghj/d+ejzp9z9xJv/2K7cXjvq9OGFwNP93pd2OXDYRYuU8VjalpnlNp8fPhxFZQRyeMbXzQiA/6Ldf/ABit/wALftZeE/A11PdaB8RYNMnuE8maRNPnfem/f/HA9e6D9iz9kXr9q050/v8A/CZP/wDHKP8Ahin9kT/n6sf/AAsH/wDjlYU8tyqnO8K07+sf8i6mbZjUVp0Itf8Abx8+/Gj9ojVf2jvhhrPgXRPGx8XahfPBMmmfY0tQyQv5jv5jwR/cx/z07V9GfBazl+HP/BNeG08Sf8Sx9O0nVWuvM/eeX/xNLj+5v/vx1e8I/sufsreCdWTVtI1C0tr1EkRH/wCEwP3HjMb/AMfvWl+0b4g8AaB+yD8RfDnhvxJpU9vHocyW1qmrpc3D750f+/l/v1305U+VYOnPmpz/AJviOCpKTaxUqPJOP/gJ8kf8L9+G/wD0OVj/AOAt1/8AGKj/AOF/fDf/AKHKy/8AAW6/+MVwv7J/7NejfF/wUmu6r4S1TWbWLxB9gub+xndIIYESB5BJs+58kjtvr7Qk/Ym/ZGW3Eyz2JgdsI48Xvsf/AMfrz6uT5VTnKnOU/d/w/wCR6aznMlCE4QhPm/xHxl8Yv2iNMtfDNq/gTxbFLrP20eb9ntZMeTsf/nvH/e2V0f7DOjeIPB/7SGlfErx1GbDRNRsL26fV5XRzI91bvsfy0y/zl/7lfUv/AAxX+yH1Waxz/wBjfJ/8XXm8dn4ePj7UfBHgW8s9bOlTTWWn6Xpt4l7dG1h/8ffYkZrrniaOW4b2WXU+d9f6ic0KU81r8+Ye5/X94k/aL+PXw7b40eIHk8V28QK2WP8ARbr/AJ9U/wBj6VxWm/Hf4c6T9l1NvGFmkkckc9rH9kuvnkR/v/6j7m8GvddN/ZN+A/irS7XWPizbLpnje5T/AEu11LxA+nSbEfy4P3PmfJ+4RKtXH7G/7JOoXBluLuzkf1l8Zv8A/F1wxy/LZv29ac1Of+E3lmOPoJUKELwgeZW//BQNTfxSS/F75UmR3H9kdt//AF6VT/4Jp/DjxP4S+N3xA1bxDpjW9tf6EWjmE0Mm8vewSfwP6A16kv7Fn7IhG0XNoFz38Y//AGyvbPBMfwe+G99dXugeJ9KguLm1S1fz/EaT/In++/8AsV6UKlLCfu8JW54T+Ln/APbTz6kZYtXrUeScPh5f/bj41+MHxq8B6P8AGHx5Z3/ie3tbq31y9gmtpLW6JjkSfDjhPY/lXHf8L9+G/wD0ONl/4C3X/wAYrzjUvANl8Zv27PibpC6dc+IrG81/XryOPS5OZtkk8kbo6de1fW/gP9iH9nO18I2I8dWi6J4oxIbyx1HxPJa3EXzvs3w5+T93seuPEZPldOryVJz/AJ/snbQzjHyoc1OEP5ftcx4ZqH7QPw+/s2+Nv4xtftX2WfyP9Fuv9Z5b+X/yw/v18zsvxO/aKt2dRJr1pojd2toBAZv++N5byff7lfpSv7Ff7IZODPZf+Fg//wAcrifiZ8O/gh+z3HpK/D/W9K06DWfO+2ibxB9r3+Ts8v7/ANw/vpK6qP1LK6FSrgFz1OnNuZ1KuMzWtCjjfcp/3T0n9qr47fD9fhr4YiPii2jeDUYUfEE8mzZZSJ/cr5f/AOF9fDn/AKHCz/8AAS6/+MV7vof7P/hnxy09p8Y9Ju9I8LwoktjPqd4+mJNc8/6t8/P+5DvsrX/4Yr/ZB/572f8A4WD/APxdeZRwuBx8Pb46c4TOuWLxOX/uMFHngeR+H/23NM8J6PDpWjfFRNP0223+TDHpckmze5d/v2n99zXnPjfUPE37SX7Qnwz8d6ZqB8Y6Xpd3p+lz6thLUb4755mQI+x/uTK+dn8dfUf/AAxX+yEw4nsh9fGD/wDxddr4J+Cf7OXw1tYrXQdasLSOC9/tGPd4sL/vv3f/AE0/6ZpXo0lh8BeeDqyb/v8AwnnupVx3uYmjyf4fiMn9vj4geHvBOo+CX8RatDpUV7LqnkSSQzOHKfZcj5Ef++K+Uf8Ahfnw4A58YWf/AICXX/xiun/4K4+LNG8UWPwhk0fWdP1ZY5Nb89rG5Sfyy/2HhynTPP5Vu/DD9h/4WW+oXEnxI8Lan4d0eSzgFtda1q72Uct0/wDBG+fnfYHfZWWKyzL6ihiK853n/Ly2901weZYyjTnQowj7n3nnX/C/vhv1/wCEysv/AAFuv/jFeK/En4teJvH3j7UPCfgLXRqegaxHBZ21rHCiLNvhQSR75kRx+839cV99H9iv9kMHmey/Dxg//wAXUHiL9mn9mPwJ4ZvvEGganp9prWi2r3mn3c/it7hIZo/9X8m/508zHyf41rg8PlmBc61L35/3uVk18dmWP5KFSHJD+6c/+wfrmmfAv9mjxd4e8eahF4d1mbWdQvUtZD9o3wPp8Cb/ANzv43wP/wB8V4pa/Hr4c29lBHL4wthJGiD/AI9br/4xXrHg3wrP8QYhLpdnN4i8M/av7N1DUtGk3wJG/l+f+/T7n7iTf/20313f/DF37Ijnc9zp0gP3ceMnH/s9cEfZZrKc8yvB/wB3/wC2O2TllMeTL3z/ANf3Twrwx+1d4T8D3VzdaD8QodMuriLyZpE06d9679/8cD10/wAevjdrnxB+GcHhXQPHknirxRqEkF6uli0SGOVIX89/n8iP7mP+elemt+xX+yCP+XizP08YP/8AF12Wg/s6/s3DVrG/sNWtv7XsIHgh/wCKukfejx7H/wCWld2Hw+Bwkb4atPm+zz/CefWxOIxbviaHukfwzd9N/wCCfdxd63a/2JdWehaq9/bh9/k/6VO+/wCSvma4/ao+GWqaC9jq3im3uLyH/j0uo7S6/wDH/wB3X1n8dtW8B/D39kv4m+GdI8RabJI3hu+S1sP7TSa4+fzPV97/ADyV+d37Jv7NOjfGDwWmv6p4T1TXLSLX/sNzfWkzxwQwIkDyB9n3MJI7ZrqxGGw+JpfWsX0/kIwuIrYef1eh/wCTHc/8L8+HH/Q42f8A4C3X/wAYrhfjJ+0Vpdt4Xs38C+LIpdZ+2fvfs9rJ/qdj9fPj/vbK+yv+GLP2Pm+5c6eE/wCxyf8A+Lof9ij9kQLxPY/j4vf/AOOVw4fA5RhaiqqUp+vKzprZpmuIh7Pk5f8AwI+WP2G9L8SeFf2kdK+J3jofY9E1GwvrmTWZJEzI88EmyTy0+f53/wBivYf2ifj58Ov+Fz+IGl8V20OVs8f6Ldf8+qf7H0pG0/w/H491bwN4DvLPW4tKnmsrDTNNvEvbryYc/wDA32YP5V3tj+yf8CPGFjbar8VIV07xxNH/AKfaX3iF7KdAnyQb4M/J+5RKxrVv7Srujj1yQh8HKa06ay2KxGCfPOe6keAW/wC0V8OrO6t7qHxrax3ELpPHJ9iuvkkSTfH/AMsK9Btf+CgKLeQSS/F/MazI7j+yP9v/AK9K9I/4Yr/ZB/5+LP8A8LB//i6b/wAMT/sh/wDP1Z/+Fga2p4DK6b/d1p/+SmVTMsfV/iYaH/gJ5d/wTR+HXinwj8cfiDq/iHTWt7bUNBkaOYTQyby97BJ/A/sah+OHxn8C6D8aPHmnan4mt7G+h1m6jmt5bW6fY+//AGEr7E8Ex/B74c3l1eeH/E+lW9xc2qWrm48QJP8Au0/66Sf7FfmP4o8Bad8af29PiXpSWU3iayutY1e7ji0qf/XbPMdHR069BXTKjTzalOeN+x/Ic1OpPLq9sND4/wCY73/hf3w3/wChysv/AAFuv/jFQah8ffh9/Z199n8Y2v2o2s/kf6Ldf6zy38v/AJYf36928B/sRfs6WvhOwTxzYronijEhvLHUvE0lrOnzvs3w5+T93seuh/4Yp/ZE/wCfqx/8K9//AI5XmwyzJ0+fnn/5KehPNsya5FRj/wCTH5rCT4o/tCRtKrXGvwaI33/MhtxAZ/8Avj7/AJP/AI5X6iftSfHDwHffDfw2v/CU2aTWuoR+cvkTfwWMm8fcryr4mfD34I/s9R6UvgLXdJ06PWfO+2ebr/2rf5Ozyx8/3P8AXPXTeHfgJ4b8YTSW/wAYNNvND0C1RLrTLu/1B9Lja6/2Js/O+ze9b4zGKvWWCUOWh3iv6juZYbDqjR+vTnzV+39e8eB/8L9+G/8A0OVl/wCAt1/8YrrtB/bW0vwnpMGlaR8VI9P06Hfshj0t32b33v8AftP75r2f/hiv9ke+kkZDZ7/vukfjBz/7UqH/AIYp/ZD/AOfiy/8ACwf/AOOVNPLcqp/BWn98f8h1M0zKrpUoRPk74mah4k/aa+NXgfxto95/wmOm6YbLS5tTIjtcOl883l7HEf8Az3T+D+OvtH9vXx94c8D6n4Jl1/VYdLhvpdUMDSQzOH2Pa/3Ef+/Wl4G+B/7Ofw5sY7PQ9b0+zt47v7ckb+Ld/wC//d/9NP8ApmlfN/8AwVi1vSvHtv8ACc6Hq+n6otmdaluXs7pJkhT/AEHG8p9K9NU6eYv6lOf7tf8AgZ58p/UmsbThaf2v5TmbD44/Du/keNPGNlGiJvd5LW62In/fin3n7RPw08v7LbeMbWO0Q4/49br5/wDb/wBRXofw8/Yr+F0GoTQeP/CmqeGPDP2JHhuta1Wayjub07P4+Pn2eZ8n+/XoA/Yr/ZEH/L1Y/wDhYP8A/HK8eOV5RKPxz/8AJT2KmbZlGekIz/8AAj4I+J3xY8Q+P/Hd74R8Ba6NW0HWoYLKC2ihRFnZ4YxJHvmjR/8AWZ64r7T/AGBdd074F/s5+J/D3jnUIfD2sza1qF8lrNvn3wPY2se/9zv43xv/AN8Vf8R/sufsy/Dfw7qninwxqWn2+v6NayX1lNJ4t8zZOnMfyZ+fmuW8F+H5viFa/atCtZvEvh37V9gvtR0d/OhRP3fnx70+5+5ffXVisd9XpwwuBp/u+r//AGTmw2E+vTnisbUtM8ptPj58OIrKGOTxja+aEQH/AEW6/wDjFbnhX9qvwj4Kvp73QPiHDpl3LB5LyR6dO+6PzN+z54H/ALn617uP2Lf2QizbbrTnT+B/+Eyf/wCOUf8ADFP7In/P1Y/+Fg//AMcrJZblVOd41p39Y/5FVM2zGorToRa/7ePnr40ftEar+0d8MdZ8C6H43bxbf3zwSppn2NLYMkL+Y7l3gj+5j/np2r6V+AWmzfD/AP4JsvaeIE/s2fTtJ1t7vf8AP5afbZ/7gerPhP8AZb/ZX8E6smr6RqNnbXyJIiP/AMJgf402P/H71v8Axy8QfDvw7+yb8SfDXh3xLpctsnhy++zWf9rpc3Du77+u/e/zvXdTlTa+p0589P8AvfEcFSUqjWKlR5Jx/wDAT46/4X78N/8AocrH/wABbr/4xUf/AAv74b/9DlZf+At1/wDGK4L9lP8AZs0j4veA217UfCWra3BHr/2Ce+spnSCCBEgkk37e4SR2z7V9nyfsXfshIG23WnFN2Ex4yc/+z151XJ8qpzlTnOd4/wCH/I9NZxmShGcIQnzf4j44+Mv7Qmk2/hS2fwJ4shl1r7Ynm/Z7WT/U7Hz/AK9P7+yug/Yb0/xH4W/aQ0n4neOR9i0TUrC+upNalkTMjzwSbJPLj+f53/2K+ppP2LP2Q9oImss/9je//wAXXm7ab4cj8eap4F8BXdprUelTTWVhpem3iXt15MP/AI++zB/KuueIo5bhPZ5bT55emv3xOaFKrmtfnzB8n9f3if8AaO+P3w6/4XJrT/8ACXW4U29n1tbr/ngn+xXmsX7QXw8huIpYPGduksLxyRv9kuvkdP8AthX0Fpv7KPwL8ZWEOp/FeJdM8bz/ACXlrfeIHsZ0RH8uD9xn5P3CR1Y/4Ys/ZBLY8+z/APCwf/4uuKOAy6u/b1pz55/4TeWYY/Dy9hRhzwgec2//AAUASO/t5Jfi98qTb3H9kdt//XpWR/wTX+Gfirwn+0B451rxDpjWltf+G7gRziaGTfI99ayfwP7GvYov2MP2RGR0EtgI2Pfxi/yf+RK9o8Iab8IfhnfTajoXifTbO4ubX7J5l34gSeN4N6f33/2K9KNSlhNMJV54T+Ln/wDbTgqOWLV61HknD4eX/wBuPjv45fGbwL4f+NHjzTtT8TW1je22s3Uc1vLa3Umxw+P+WaVw/wDwv34b/wDQ5WX/AIC3X/xiuD8UeA9P+NX7e3xL0lLGfxLZXmr6xdomkz7/ADAnmOjo6deQK+qPAP7EP7O9v4Rsh4+sV0LxQXmNzYal4mktbhE8yTyPMhz8n7vy65K+UZbTr8tWc9fe+zy/kdtDN8fKhzU4Q0937XMeH3Hx9+HX2W78nxja+f5Enkf6Ldf6zy/k/wCWH9+vmhk+J/7R0OVD+ILbRW43PbwLAZvrszu8n3+5X6VD9h/9kfyfOMtn5f8Az0/4TF/8a4z4m/Df4H/s/LpUXw/1zStOi1vzvtvneIEu9/k7PL++/wAn+ukrqo/Usqozq4Bc9TpzGU6uJzatCjjvcp/3T3/9r7/kW/CP/YTm/wDRBr5jr6c/a+/5Fvwj/wBhOb/0Qa+Y6+IzT/eT63J/91CiiivIuz1z5m/bU5fwSP8Apjdf+hpX6e/tnHOj+HP+wtc/+iDX5h/tqcN4JP8A0xuv/Q0r9Cf+CknxKT4UeDvCGoXOjy6nHPrV1HsW4Fv/AMsPUo/rX6NTw9TE5LQp0Pj97/0o/PKlaGHzqdSt/Xunh1cX8a/+SPeNP+weP/SiOus8P3M/iDQNL1VLOaGO/s4b1FH7wJvTfs3/APA65f4328kfwa8abo5E/wCJePvp/wBPEdfD4OMqWOhTktpx/wDSj7TEuLws5r+SR7p/wTlb7R+wz4oilf5VvtajSST+AfZYP8a4C30ufzI7XZ+82V3H/BOGPzv2G/Fka7M/2lrSff8A+nK3rmbfUIP7Nj0vz/3ez/j7/wA/8s6+l4j/AN5PmeG/4FQzbiNI55EhfzI/+elQVPcW72c+yb/WVFXx92fanF/GiPy/gz41/wCwf/7cR17v/wAE3Vz+xJr/AP2Eta/9JIK8I+Nn/JGfGv8A2Dx/6Pjr27/gnrqbaP8AsF+M9Q+ztOtjd69P5afx7LGN/v199kcHUyusv758Dn0lTx9G/wDJ/wDJHn9n/wAekH+5HU1cJ8G/idH8WtF1K5tdGm0z+y3gg8uS68/fvST/AGE/5516B9jn/wCeM3/fFfE18PVwdX2Ff4z7ejXp16ftqPwHF/8ABPv/AJP4+Iw/6ctc/wDSmOvdf2hv+S0eKPra/wDpJBXhf/BPz/k/74jf9emuf+lMde6ftDf8lo8UfW1/9JIK+04k/wB3o2/u/kfE8P8A+/Vv+3v/AEo85ooor8/uz70K5D9gH/k/b4g/9eeu/wDpQldfXI/8E/1b/hvf4g7VZ/8AQ9dHyf8AXwlfccNX5K/+FHxnEnw0z3P9oRt3xo8UH3tf/SWCvOqn/aI+LMcP7Z3iL4d/2HNJeTS2Kf2j9q9dOgk/1Gz/ANnpv2Of/njN/wB8V8/mOFrYWtL26/iao9/Lq9PEYSHst4HzD+2p97wV/wBcbr/0NK/UL9s7/kWdC/7DT/8AoiSvzB/bYjkjk8Fb45I/3N1/rP8AfSv0+/bO/wCRZ0L/ALDb/wDoiSvpammRUP8At7/0o+W/5nkv+3f/AEk+V45Ht5I3R/LkT/lpV+SNNUjknhTy50+/bx/x/wDTSOs+rOn288k++F/L2fvHuP7lfCxufe2Czs3vJNifu40/ePJJ/BXzZ+3BdIy+Cba23R2ohuu/Mn7xPnevqW8kTVIZEsP3ez949v8A89/+mn/2uvkz9tTr4K/643X/AKGlfVcP+7mdL5/+knzWee9l8/6+0fqB+2Z/yLeh/wDYaf8A9ESV8r17x/wUe+JEfwp+HXhjU7nR5tSjm8QPCY0n8j/l1k/j2PXz/wCHrqfxF4f0vVks5oY7+ygvo0Hz7N6b9m//AIHXPm+FrU5/WZL92x5FiKdSh7CPxnJ/Gz/kjPjX/sHj/wBHx17x/wAE2f8AkyPX/wDsJa1/6SQV4X8b7eSP4N+NN0cif8S8ffT/AKeI691/4Js8fsR6+Dwf7S1r/wBJIK9zJP8AkV1r/wA542e/7/Rt/L/8keaWf/HjB/uR1NUNn/x4wf7kdTV+ftu59/FKyCuL+Nn/ACRnxr/2D/8A2vHXaVxfxs/5Iz41/wCweP8A0fHXXgW/rVH/ABR/9KODG/7tW/wSPd/+CbbY/Yo8Qf8AYW1kf+SsFebWf/HjB/uR16B/wT11BtJ/YL8Z6h9nadLG71248tP49ljG/wB+vC/g38TI/ixomo3Vro02mjTHgg2Pdefv3pI/9xP+edfZ8SYTEVZvEL4IHyPDeJo04+xfxndV59+wj/ykM8W/9c9e/wDQ2r0b7HP/AM8Zv++K85/YR/5SHeLf93Xf/Q2rHhm6+sX/AJUdPE3wUbfzH0N+0J/yWjxR9bb/ANJYK86r0X9oT/ktHij623/pLBXnVfJ4p/v5n1GD/wB3h/gCrlneJHH9luf3lo//AI5/00jqnRXMr6HT0OK/Yc057P8A4KFeJy3z280WvNA6fcdA719C/HiyT/hcnim6uf3dohtv+B/6LB+7r5//AGDVum/by8Ws6O9tjxBsx03h/n2e/wByvQP2lfi0bf8AbG1v4eS6Dc+dNLapDqH2r93j7DHJnyNnv/fr9NzvCVMRh4zpxu4QPznJ69PD46ftJ/H/APJEF5ePeSfP+7jT92kcf8FfLv7axIPgnHH7m6/9DSvp77HP/wA8Zv8AvivmT9tmKRZPBO6ORD5N19//AH0r5Xhrm/tOF/690+mz639mz5f694/Tn9sH/kU/DP8A2Ex/6SmvlyvqP9sb/kUvDP8A2Ex/6SmvlyuHOP8AejbJv90gFFFFePdntnzN+2ocN4K/64XX/oaV+nn7YH/Io+GP+wp/7amvzD/bV6+Cf+uN1/6Glfod/wAFFPiNH8LPhj4Q1K50eXUo5Nb8jYtwLf8A5cpP9h/Wv0ahh6mJyWhTp/H73/pR+e1K0MPnU6k9vd/9JPCq4z41cfBvxp/14D/0fHXWaHdT+INA0vVUs5oY7+zhvUUfOE3pv2b/APgdcr8crd4/gz413xyJ/wAS+P76f9N46+HwcZUsdCnL+eP/AKUfaYlp4Wc1/JI9l/4Jkf8AJlnjT/sOat/6brSuFs/+PGD/AHI67r/gmR/yZb40/wCw5q3/AKbrSuFs/wDjxg/3I6+l4m/3nQ+a4Z/gO/8AW5NRRRXxl2fZnFfG/wD5Iv43/wCwdH/6Pgr2v/gmUuf2L/GX/Yd1r/03WleKfG7/AJIv42/7B0f/AKPgr2P/AIJzak2jfsI/EXUzA062Opa7P5afx7NLt3/+tX6DkUHUyqtD+/8A/InwGfyUMfRn/c/+SOW0e8+z2MaOnmQOkfmR1JeWf2fy3R/Mgf7kled/Bv4nR/FnRdSubbRptM/st4IJEe68/fvST/YTZ/q69W0u3fT4JJ7y1eS0m/5dNn3/APppXxtfD1cNV9hX+M+0o16den7aj8B5r+wzZSQ/t/eKricCNJk17yY+z8n/AMcxXun7RH7z41eJf962/wDSWCvEP2NIJR/wUO8UXUkqXMMya8I5en3M/u/9jZxXt/7Qv/JZ/Ev1tv8A0lgr7LiJ/wCzUX/hPjMj/wB+rf19o86ooor8+uz76wV5/wDsN/8AKRDxb/1z17/2evQK4b9htWb/AIKGeLdqs/7vXvufU19vw1e+IX90+N4k/gw/xHv37QjbvjR4oPva/wDpLBXnVSftG/F6O3/bL174d/2FPJdzT2MH277V/wA9LGB/9Rs9/wC/S/Y55I/9Q/8A3xXz2ZYarhq375fGe9l1enXwsPZbwPmP9tP/AI9/BX0vP/aNfp3+2D/yJ/hj/sKf+2slfmR+2xFIkXgsNHIhxecP/wBsa/Tf9sL/AJE/wx/2FP8A21kr6h6ZFQ/7e/8ASj5r/meS/wC3f/ST5booor4C7PurahXzR+2p/wAe3gz63v8A7Rr6Xr5o/bU/49vBn1vf/aNfS8Nt/wBp0fn/AOknz2f/APIvq/19o/Tv9sL/AJFLwz/2FP8A21NfLde6/wDBRT4jR/Cz4Z+ENTudHl1NH1vyPLS4Fv8A8uMn+w/rXgXh+6m8ReG9L1VLOaGO/s4b6OEfOF3pv2b6M4wtanP6xJfuzLJK9OeH9hH4zlvjL/ySDxt/2D//AGpHXt//AATJb7R+xf4simf5V1jVkTf/AAf6Db/414r8ZLeeP4Q+Nd8Dx/8AEs/uf9NI69l/4Jhxmb9jLxpGu3/kOav/AOm60r3Mk/5Fda/854+e/wC/0bfynF2elz/uLXZ+82R0XEaRzyJC/mR/89K0rPUIP7Kg0vz/AN3sj/0v/P8ABWbcW72c+yb/AFlfB1Nz7dJWIK5H4wx+X8IPGv8A2DP/AGpHXYVx/wAZP+SQ+Nf+wZ/7UjrpwDf1un/jiZ4pL6rP/BI9l/4JlPj9i7xt/wBhzWf/AE3WlcJZ/wDHjB/uR12H/BOPUm0n9hP4i6n5DTpY6rrlxsX+PZpdu+P0xXi/wX+KEfxd0rVXttGm0z+yfssEm+68/wA7ekg/uJs/1dfYcRYTEVZTrr4IHxvDmKo017F/HP8A4J31cH+w3/ykM8Wf9c9e/ma9B+xz/wDPGb/vivPP2HP+UiHiw/7Ovf8As9Y8M/8AL+/8p1cSfwYW/mPoP9oT/ktHij623/pLBXnVei/tCf8AJaPFH1tv/SWCvOq+UxTft5n1GDS+rw/wBRRRXGr3OnocL+wr/wApCvFv/XPXv619BftFf8lm8RfSz/8ASVK8A/YVVn/4KEeLtqs/y679z6mu/wD2lvi9Hbftla18PP7DmkvJ20+3+3faePnsbeT/AFGz3/v1+mZ5ha2Kw8fZfYgfnOU14YfHz9p9r/5Ir180ftpEi38GfW9/9o19OfY55P8Alg//AHxXzP8AtrRSJB4KDRyIcXn3/wDtjXzHDaf9p0b+f/pB9Nn1v7Oq8v8AXvH6e/tg/wDIj6B/2F4//SWSvlaOR7eSN0fy5E/5aV9U/tg/8iPoH/YXj/8ASWSvlWubN3/tRrkv+6I0JI01SOSeFPLnT79vH/H/ANNI6gs7N7yTYn7uNP3jySfwUafbzyT74X8vZ+8e4/uVpXkiapDIlh+72fvHt/8Anv8A9NP/ALXXk/xD2D5c/beug1r4LtrbdHag3uMnmT/UfO9fpn+19/yIvh//ALDCf+kslfl7+2p/x7+C/re/+0K/RT/goj8Q4/hX8IfC+s3OlTalG/iJLUwJcCDrYz/x7Hr9Ap4epjMmoU6fx+9/6UfAV60MPnM5z293/wBJPAq4/wCMn/JIfGv/AGDx/wCjI66nw/dTeIPDukaqlnNDHf2cN7HCP3gXem/Z5lc18YreeP4Q+NN8Dx/8Swfwf9NI6+Iw0ZUsXCnLpOP/AKUfcYpxqYWc1/Ie0f8ABLv/AJM78Wf9jBq//putK890/wD48bT/AK4JXof/AAS34/Y78WZ4/wCKg1f/ANN1pXnmn/8AHjaf9cEr6Tib/edD5jhn+HUv/W5NRRRXxl2fZWOP+Mn/ACSHxr/2Dx/6Mjr2z/glv/yaB4w/7GLV/wD03WleJ/GT/kkPjX/sHj/0ZHXs3/BMu5bSf2LfHd55TTpa63rVx5afx7NOtK/QcivUy2sv758Dn37vG0f8Jwen/wDHjaf9cEqauB+C/wAUI/izpWova6HNpv8AZP2WCTfdef529JP9hNn+rr0P7HP/AM8Zv++K+JxGHq4Or7Cv8Z9vRr08VT9tR+A8+/YZ/wCUhni3/uPf1r6F/aM/5LR4i/3bL/0lSvnr9hn/AJSGeLT2xr39a+hf2jP+S0eIv92y/wDSVK+04j/3ajb+7+R8RkX+/wBa/wDe/wDSjzaiiivz+7Pvi5Z3iRx/Zbn95aP/AOOf9NI681/Y3017H/goxqu/EkEs+vNHIn3H+SSu+riP2IY5/wDh4FrvmpvtftPiHZhP49j79n6V93wxduv/AIT4niSyhB/3j6L/AGgLOP8A4XJ4iurn93aItl/wP/RU/d15peXj3knz/u40/dpHH/BVn9pr4tm2/bC1HwA+hXDS3DafHDqP2r9389lG/wDqNnqT/H2ql9jnk/5YP/3xXzmaYerg68nP7Z72V16eIwsOX7B8y/tpf8e/gr/dvf8A2jX6cftXf8kx8If9hGH/ANIZK/Mj9tSN0i8Fb43Q7b3/AFn/AGxr9N/2rv8AkmPhD/sIw/8ApDJX1T/5EVP/ALe/9KPmP+Z7L/t3/wBJPlyiiivz27Pv7BXzV+2j/qvBv0vf/aNfStfNX7aP+r8G/S9/9o19Lw23/adH5/8ApJ89xB/yLKv9faP02/aq/wCSY+EP+whB/wCkMlfL1e5/t+fESP4V/AvwZrFxpU2po+r2tqY45/I/5h0n8ex68E8P3U3iHw7o+qpZzQR39lDexwj59u9N+zf/AB1pnWFrU5/WJL92YZDiKc6HsI/Gcx8ZP+SQ+Nf+wYP/AEZHXs3/AASx/wCTP/G3/Yxap/6brSvHPjJbzx/CHxrvgeP/AIln9z/ppHXsn/BLHj9j7xtn/oY9U/8ATdaV7GSX/sutf+Y8vPP9/o2/lOB0/wD48bT/AK4JU1Q6f/x42n/XBKmr8/bdz75LQK474zf8kg8bf9g//wBqR12Ncd8Zv+SP+Nv+wf8A+1I67cA39bp/44nJi/8Adan+CR7P/wAEtG/4w+8bHH/Mx6sP/KXaVxWh3n2exgjdPMgeCPfHXX/8EwZnsv2LfiDeeTJOlrr+ry7E9tKtDXiPwX+KEfxc0rUXttGm0z+yfssEm+68/wA7ekn+wmz/AFdfZ8SYfEzm8RD4IbnxPDlejFexqfFP/gno95Z/Z/LdH8yB/uSV5x+xXp7Q/wDBQbV7icCMTT+IfJj7SfJJ/wCOcmvWdLt30+CSe8tXktJv+XTZ9/8A6aV5b+yKsw/4KNapdPKkyTT68Ek6fchk/d/7GzisOGW066f8p08R/wAOFv5j6A/aMk3/ABr8Rf7tl/6SpXm1ek/tGf8AJaPEX+5Zf+ksFebV8jir+3n/AIz6jAf7rT/wBRRRXJrc7eh5v+xP/wApG9bP/TbXv/Rc9fR/7RX/ACWbxF9LP/0lSvnL9idWk/4KKa4ERnPn699z/rnLXpv7Svxfjtv2zNa+Hn9hzyXc7afb/bvtPCb7G3k/1Gz3/v1+nZ5hK+KwsXS+xC5+c5TiqOHx0/afa/8AkiCvm39tAn7H4J/66Xv/ALQr6X+xzyf8sJv++K+av204pEsvBW6ORP3moff/AO2FfK8OJ/2nRv8A3v8A0k+qz5r+za1v694/TD9q/wD5Jn4Q/wCv+D/0hkr5fr6g/av/AOSY+EP+vyD/ANIZK+X6583/AN6Fkf8AuUAooorw7s94+a/20v8Ajz8E/wDXS+/9oV+mf7Vn/JMfCH/YRh/9IZK/Mz9tP/jz8E/9dL7/ANoV+g/7ffxDj+FfwL8G6xc6PLqaNrdrbGNLjyOunSfx7Hr9Hw+HqYrJqFOlv73/AKUfnmIrQw+cznPb3f8A0k8MrmPil/ySvxn/ANgmb+dbfhu+n8SeGNH1lLKaBNStkukg+/s3/wC3/HWP8VreeP4VeM98Dx/8Smb+D3r4jCRlSxtOm/5z7iu1Uwk5r+Q9i/4JYs9x+x/4wWZ/kXxHqkaF/wCD/iXWlcFp+lz+XaWuz955Ef8A6LruP+CV8Zk/ZD8ZIf8AoZtT/wDTbaVzWn6hD/Y9ppfn/u/Ij/0v/P8ABX0/EemJPleGf4dS5QuI0jnkSF/Mj/56VBU9xbvZz7Jv9ZUVfF3Z9r0OX+KUePhT40P/AFCZq9e/4JY/8mf+Nv8AsYtW/wDTXaV5D8U/+SU+NP8AsEzV6x/wS/uHsf2LfiDeiF50tdf1iTYn/YKtK+/yK9TLay/v/wCR8DxD+7xtH/AcTp//AB42n/XBKmrz74L/ABUj+Lmm6r9i0KfTP7J+yo/mXP2vfvSQf3E2f6uvRvsc/l/6ib/viviMRh6uDr+wr/Gfb0K9PFU/bUfgPOP2N/8AlI5ro/6aa9/6Ikr6L/aO/wCS0eIf92y/9JUr51/Y5U/8PHtbXt52vf8AoiSvor9o7/ktHiH/AHbL/wBJUr7TiD/c6P8A27/6SfEZH/yMa1/73/pR5tRRRX5/dn34V57+xx/ykg1r/e1v/wBEyV6FXnv7GqO//BSPWdqO/wA2t/c/64yV9vwxd1K/+A+M4k/g0/8AGfRf7Rhz8a/EP/XKy/8ASVK82qT9pj4ywWP7ZWt/D86BPJeTS6Xai++2cIJLKB/9Rs/6af36X7HP/wA8Zv8AvivBzLB1sLXk66/ianvZXXp18LD2W8D5n/bWP+h+Cf8Arpff+0K/TT9qz/kmfg//AK/4P/SF6/Mv9tWGRLPwTujkT95fff8A+2Ffpp+1Z/yTLwf/ANfsP/pC9fTvTIqX/b3/AKUfKv8A5Hk/6+yfL8cj28kbo/lyJ/y0q/JGmqRyTwp5c6fft4/4/wDppHWfVnT7eeSffC/l7P3j3H9yvgos++sFnZveSbE/dxp+8eST+CvnD9tu6H9m+Cba33R2ge+PXBkf9x8719Q3kiapDIlh+72fvHt/+e//AE0/+118o/tpf8ePgr/rpe/+0K+m4e93M6S9f/ST5rPPey+t/X2j9N/2rP8Akl/hn/sIWv8A6Qz18t173+3/APECP4U/ATwnrV1pM2pxvrdla+R5/kf8uM/8ex/+edfPHhq+n8S+GNH1lLKaCPUrZLpIPv7N/wDt/wAdGdYWrTn9Ykv3ZGQ16c6HsI/GYfxT/wCSU+NP+wTNXsf/AASn5/ZH8XZ/6GfVP/TdY15B8VreeP4U+NN8Dx/8Smb+CvX/APglPx+yP4vz/wBDPqn/AKbrGvWyP/kWV7/zf5HlZ5/v1C38p59p/wDyDrT/AK4R1NUOn/8AIOtP+uEdTV8C27n36Ssgrmfih/ySvxl/2Cpv5101cx8Uf+SV+Mv+wVN/Ou3Bt/WKf+OJxYz/AHeoexf8EqX/AOMR/Fx/6mjVB/5TrGvPtP8A+Qdaf9cI673/AIJY74f2OvGs/kvOkPiXVJCie2l2leD/AAX+KifFzTdV+x6FNpn9k/ZY333Xn796SD+4mz/V19lxJhK1adTEQ+GB8Zw1iqNNOg/imeg157+xr/yke1r/AH9e/wDRMlej/Y5/L/1E/wD3xXnP7HKn/h49rKkcedr/AP6Jkrm4ZunXv/KdnEn8GFv5j6N/aU/5LRrX/XtZf+iUrzOvTP2lP+S0a1/17WX/AKJSvM6+Uxv+9TPqMB/ulP8AwBVyzvEjjktbn95aP/45/wBNI6p0Vzo6ehxP7IdlJY/8FItY34kjb+22jeP7j/uJK+j/ANoayQfGLW7q5/d2iW9n/wAD/cp+7r54/YtWY/8ABQnXAyB7XdrmzH9/yfn2e/3P0r0P9qr4tLpv7YE3gGXRLmSa8XT4Yb37b+7TzLWP/lhs9/7/AGr9MzbC1cVg6Xs4XcIRZ+c5TiaeHzKq6k/i/wDkineXj3knz/u40/dpHH/BXzF+2kcWfgr/AK6X3/tCvpn7JP8A88H/AO+K+av21opEtPBIeORP3l79/wD7YV8tw/J/2pRv/e/9JPps7t/Zta39e8fpZ+19/wAi34R/7Cc3/og18x19Oftgf8iv4S/7CU3/AKINfMdefmn+8nVk/wDuv3hRRRXkansanzP+2ofm8Ff9cbr/ANDSvdPGH/BNb9ozxlDHZeJ/ix4d1+C2ffHDq/iHVLpEcjl032p/SvCf21vveCf+uN1/6Glfqn+1Z4w13wdpuhS6Bq1zpE9xqk0M0lqI/nTyC/8ASv1PBY2WAyihO383/pR+YY3B/Xs3nRX9e6fDFv8A8Ey/2grOFIIvih4Xt7eJNqJHrmpoiJ6D/ReBRefsW/Fj4F6fe+PfHfjbQfE3hHQ4nu9T0m11O8upLmH7nEM8EaP87pw7ivZ7f4vfEa8n2Q+MtV3/AO9H/wDG64j9on4seLdW+BXjfSLvxTqGp2D2A87zgn779/H/ALH3K5qWe0cXNUJw+N8p11MlxGFp+2jP4D3n9lvxN4f8XfspeMLvwtbPZ6ZHPrMDf6Glrvk+zIfM8tP+ugrwK3/490/3K77/AIJ0n/jBfxZ/1/69/wCkVvXA2/8Ax7p/uV4fEVGNCtCED1+Hasq1OpOX9bmlb3CXEH2W5f8Ad/8ALC4/uf8A2FVLi3ezn2Tf6yoq0Le4S4g+y3L/ALv/AJYXH9z/AOwr5T4z6w88+Nf/ACRrxp/14D/0fHXDfsw/ssfG/wCNnwhn1bwL8TLPwz4TmvbqwfSbrW7+1RpPLTz8wQwOmHSRPrXd/Ha3ez+D/jaOb7/9nx/+j4695/4Jo3Ulj+xjrM8Duk8Or6vOkifwSJaQbK/Q+Hq88Pl1ap/f/wAj894hj7bG04f3f/kj570f/glh8dNBjdNM+Ivg7T1mG90s9X1BN/8A3xaVoL/wTQ/aFGM/FXwyf+47qn/yLXrVv8aviHJaxyf8JtrHzon8Uf8A8bqT/hdfxD/6HbVf++o//jdTLiWE38H/AJKX/q9iKaup/iUf2L/EWjaf+0dqvgFrP/itfDOlahpmralFYwbLq5tpIYZ5I58ec+949/zpl8/PXQftBDHxm8Tj3tv/AElgrxX9ge8n1T/goH8S7y7mae5mttemluG/id7tMv8Amc17X+0J/wAlo8UfW2/9JYKx4iowo0oOBrklSdXFy597HnVFFFfA6n3eoV81fDL4T+P/AIw/tUeOtE+HHilfB/iBLrU7t9Qkv7qyHkJOfMTzIEd+rJxX0rXJfsA8ft7/ABB/68td/wDShK+84Xn7N16n91HxnE8OelTIL7/gl/8AHvVPEUniC9+JfhK81xxiTUp9b1B7s/Js5f7JvPyfJVhf+CZ/7Qw/5qp4bP8A3HdU/wDkWvpb42fFDxn4f+KXiDTdK8U6jpunQGEQWsDR7E3wI/8Azz/vvXD/APC6viF/0O2q/wDfUf8A8brsq8TQjU9nOH4Hm4fIKsqaqU57nz947+G+ofse6bBZfGC807xvd+IvMfTJtNX+1BapBlJ43+1pHs3vJH9z/nnX3v8Atkjb4V8Oj/qNN/6Ikr82v28vGfiDxY/gX+3NYuNXkhhvDCbraCgeRPT6D8q/S39sSzkvPD2hon7uNNad3kk/gTyJKeYKniMCsVD7f/toYWVSjmMMNU+wfLFnZveSbE/dxp+8eST+Cn3l4nl/Zbb93aJ/4/8A9NKLy8Ty/stt+7tE/wDH/wDppVOvzk/RRY5Ht5I3R/LkT/lpXzx+2pAurr4RmgjWOeG1upJoR/Gm9PnSvoavmr9tCV7e48EvE7IyQ3Wx0/30r6Xhv/kZw/r7J89xB/yL5/19o9x8Yf8ABNX9pDxlDHp/iT4ueH9fghm3rDqfiDVLpEc9X+e1I/Gqtr/wTK/aAsoUhh+KPhe3ghTaiR63qaRonp/x6cCvvL9qPxFr3hnQdOuPDurXOkXL6m6TSWrJvdPIkevm3/hdXxC/6HbVf++o/wD43X1GKz/6vP2c4HymByOrXp+3pzPHNS/Yx+KXwJ0+7+IPj3xroXiTwdoKfatT0m01G9upLmHITYsM8CRv88iffcV9Yfsf+JtE8Y/st+J9R0CzawsTe6zH5bW8dt+8+yofuR8d6+d/j/8AFXxnrvwN8cWmo+KdS1Cxm0/ZJBOE2P8Av4/9ivTf+Cbylf2J/EP/AGEta/8ASWCnTrUsww88ZDScfdJxFOvgasMHVn8XvHmtn/x4wf7kdTVDZ/8AHpB/uR1NX5c73P1CPQK4v42f8kZ8a/8AYP8A/a8ddpXF/Gz/AJIz41/7B4/9Hx12YH/eqP8Aij/6UcGN/wB2rf4JHDfswfsr/HH42fCObVvAvxMs/DPhOW9urGTSbrWb+1R5PLTzswQwOmHSRPrXY6P/AMEsPjnoCsml/ETwfYJMN7pZ6vqCb/rstOa+hP8AgmlcS2f7F+tTwO8c8OsatIkifwSJawbKxbf41fEKSGN/+E21T50/vR//ABuv1DM87+oVvZzWh+aZZlLx0eenM8l/4doftC7gT8VfDH0/tzVP/kSvR/2PfEmg6T+0o/gGayEvjbw7YahpGp6lDYwbLm6tlSOeSOfiZ97xufnT58jfWv8A8Lq+IX/Q7ap/31H/APG68X/YevJtU/4KJeMrm6maa5mbxBNLO38Tu75f8Sf1rnwuNo5vGp7luQ68Xha+VQtKfPznvf7Qn/JaPFH1tv8A0lgrzqvRf2hP+S0eKPrbf+ksFedV+b4r+PM/QsH/ALvT/wAAVcs7NPL+1XP7u0T/AMf/AOmdFnZp5f2q5/d2if8Aj/8A0zpl5ePeSb3/AHcafu0jj/grnjujpex8weCPh549+M37VvizQfh/4ng8Ia4t1qkkN619PZRpAszl4/MgR3+fPTHz9692X/gmr8ebvWrjU9V+JHhXUr+6dHk1CfWtRe7DogSN1m+y79+wbP8Ac/DGD+wi3/GwjxZ/u69/6G9fVXxq+KXjDw98U9d0zSfFOpaZp0Jh8i1gaPy03wI7/wDLP++9fruPzP8As+lFf3T8mw2WvHYicKZ81N/wTN/aGiUhfip4aKf9h3VP/kSuY8ffDfUv2PdLgsvjDe6d43vPE3mSaXPpi/2oLVIPknjf7XHHs3vJH9z/AJ519FWfxw8exybJvGGrSQSf7cfmJ/5Dr5Q/b+8TeIfEz+BZNc1m41uNIbwWtxcbf+eib+n0Fedgs0o5tUWFnC3N202PTxGXYrK6f1lz+E/Rn9sL/kUvDH/YTH/pJJXy3X1H+2N/yKXhn/sJj/0lNfLlfD5urYqx9bk75sLBhRRRXj6ntanzL+2t/wAyV/1xuv8A0NK918X/APBNn9ozxxYwWXiX4s+H9fs4X86CHVvEGqXSK+z76B7U4+T0rwv9tX73gr/rhdf+hpX6o/tQeMNc8IeGvD9xoerXGkzy6j5M72ozvT7KX2fpX6ngsdLAZRQqW/m/9KPzHHYP69m06K/u/wDpJ8MWv/BMv4/2NukEPxQ8LwQRJtRY9c1NERPT/j06U3UP2Mfil8B9NvPiB4+8a6D4m8H6Cv2rU9LtNRvb17mHITYkM8CRv88iffcCvY/+F1fEL/odtU/76j/+N1wnx++K3jPXPgb44s9S8U6lf2E2nhHt5xH5b/v4P9iuOjn9HF1Fh5w+PQ76mTYjCw9vGfwHv37G/ivQvG/7MfjLUPDlm2n2SahqsDo9nBa/P9igO/y4+P8AloleCWf/AB4wf7kdd5/wTJP/ABhb40/7Dur/APputK4Oz/49IP8AcjrxuIKMKNeFGB6fDtWVaNStImooor5E+t1OK+N3/JF/G/8A2Do//R8FcF+zB+y/8bPjd8Jb3U/AfxGtPDPhSTUbnTZ9IutZv7VJpjDH52YYYXQ745EX3xiu9+N3/JF/G/8A2Do//R8Fe7/8EwpJ9J/Y58UXvzRyLr2rXEEn+5YWnz/99pX6Tw/X+r5dVqQ/n/yPzziGHtMbTh/d/wDkjwzwz/wS0+Nvhkv9n8f+Eba3k+fyLTWtRhSVv9vZa1buv+Cbf7RN1O003xX8Ms79f+J7qn/yLXrEfxv+IdxBG7+NtV8x08z70f8A8bo/4XV8Qv8AodtV/wC+o/8A43WcuJoSlfk/Aunw7iKa0n+Jm/sc+KtH8P8A7SU/w3mtfP8AGWg2GoaXquoW9nH5Nxc2qok8iT8TPveN/nkSuo/aIs3j+MfiGb+BzbbJP+3WCvBv2J7661L/AIKLeM7q8ne4u5hr8jTv993ya99+O995Pxm8Uo8fmQObbfH/ANusFYZ9RpUMPDl+2VktSrWxk+feB5nRVm8s/s/luj+ZA/3JKrV8IffahXzL4B+Gfjn4s/taeM9A+HnidPCHiJrzVJxqMt7PZDyUkcyJvgR36dq+mq4D9hn/AJSH+Lf+uevf+z19zwvP2br1P7qPjOJ/4NP/ABFzUv8Agl/8e9Y8QNr198TPCN5rjBHfUp9a1F7v5E2J+8+y+Z9zH4Cp1/4Jn/tDD/mqnhs/9x3VP/kWvpb42fFDxn4f+KWu6ZpXinUdN0+AwiG1gaPYm+BH/wCef9964f8A4XV8Qh/zO2q/99R//G67KvE0Iz9nOH4Hm4fIasqftKc9z5+8ffC3Vf2OtNgh+MGoaf44n8UbzpEmmn+1PsqQDZPG/wBqRNm/z4fuf886+8/2w/8AkUfDB/6if/tpJX5xft5eOPEPjCHwJ/bmtXerfZ1vfJa725Tf5BccfQflX6PftjHPhHwx/wBhT/21kqcwlDE5csbT+3/7aaYP21DMIYOp9j/24+WqKKK/ONT9C1uFfNH7an/Ht4M+t7/7Rr6Xr5o/bU/49vBn1vf/AGjX03Df/Izo/P8A9JPnc/8A+RfV/r7R7l4u/wCCbP7RvjaxgsfEnxZ8PeILOF/Ogh1TxBql0ivs++m+1P8AB6VUtf8AgmT+0BYwJDF8UPC8EEKbUSPWtTRET0/49OBX3R+1B4w13wf4Y0C40PVrnSZptREEz2o++n2UvsrwC3+L3xGvJ9kPjLVd/wDvR/8Axuvq8Vn/ANXqexnA+TwOSVcRT9vTmeM3n7EXxZ+C9neePPG3jbQvEvhLQInv9V0i11K9upLm3T76eTNAkb/f/jcV9Vfsh+L/AA54v/Zj8YXfhW3e00+HUtWgk8yyjtd832KOTzPLT2kSvCPj98WPFmqfA/xxpdz4o1LULN9MdJ/PEf7/APeR/wCx9yuz/wCCZfP7F/jQf9RvWv8A03WlVTrUsww88VDSUfd8jPEU6+BrQwtT4Ze8cBZ/8eMH+5HWtb3CXEH2W5f93/ywuP7n/wBhWTZ/8ekH+5HU1fmMdz9QV7EtxbvZz7Jv9ZXFfGT/AJJD41/7Bg/9GR16Bb3CXEH2W5f93/ywuP7n/wBhXDfG63ez+E/jZJv9Z/Zf/tSOu3Cf71Q/xxOPFf7rP/BI81/Zb/Zd+Nnxs+Euoar4C+I1n4Y8KtqF1ptzo91rN/aiaYwx+dmCCB4zvjkROevSu30f/glh8c/D6Oml/ETwfp6TffSz1fUE3/XZac17r/wTDuJrP9jnxhcQO6Tw6/q0iSJ/BImnWmyqlv8AGr4hyWscn/Ca6r86f3o//jdfp2ZZ0sDV9nNaH5pluVPHR56czyQf8Ez/ANoYNn/hanhvHp/buqf/ACLXpH7HmvaJof7TMnw8mskk8b+HtN1HS9T1KKyh2XF1AqJNIlx/rpN7x/xpWv8A8Lr+If8A0O2q/wDfUf8A8brxr9inULnVP+Ci/jG5u5nnvJm153nf77vlq5MLjaWbqp7luQ68VhK+Vw96fxnvH7Qn/JaPFH1tv/SWCvOq9F/aE/5LR4o+tt/6SwV51X5viv48z9Cwf+70/wDAFFFFcavc6Xex8z/D/wCGfjj4tftZeMtB+HnihPCHiI3uqT/2hJqE9kBCkjmRN8CO/SvZdW/4Jg/H3WPEJ1y/+JfhW71z5P8AiZT6zqL3fyJsT959l3/cwPoKg/YX/wCUhXiv/rlr38zX1T8cvid4w8P/ABX1rTNJ8TX+madAtqI7SBo/LTfAjv8A8s6/YcZmn1ClHT7MT8lw2AeOxU6aPmMf8Ez/ANoYLj/hanhvPr/buqf/ACLXOePvhhq37HGm28Pxh1DT/HFx4o8w6PLpZ/tT7KkA2Txv9qRNm/z4fuf886+gv+F1fEL/AKHbVf8AvqP/AON18qft5+OPEHjC38C/25rV3q32b7b5Bu9uU3+QXHH0H5V52DzSjmtdYOcLc3bQ9evl2Kyun9bjP4T9JP2vv+RD8P8A/YXj/wDSWSvl2zs3vJNifu40/ePJJ/BX1T+1tZveeCvD6J+7jTV43eST+BPsslfLN5eJ5f2W2/d2if8Aj/8A00r4vNFbFWPpsnd8LcLy8Ty/stt+7tE/8f8A+mlVI5Ht5I3R/LkT/lpSUV4x7x8+/tqWq6va+EpreNY7iFL2SaEdHT9x86V694s/4Jt/tHeNLGGw8R/Fvw/r9jFIsyW+peINTukjfZw4SS1OPkrw/wDbMle3XwWyOySKb350/wC2Nfqv+0x4l1zw34P0a78O6nc6TfSaiiTyWoTe8f2WR6/U8FjZ4PKKFT/F/wClH5hjsH9azadH/D/6SfBtt/wTJ/aAsrdYIfij4Xt4Ik2okeuamiIn/gJwKdffsT/Ff4IaTeePfHXjjQvEXg/QYvtup6TZ6ne3clzBwmxIZ4ER/vpw7ivX/wDhdXxC/wCh21T/AL6j/wDjdcR8cPix411r4K+OrLU/FOpX9nNprxyQTtH5b/vI/wDYrgpZ/RxdRYedP49D0HkeJwsPbwn8Hme//sT+LND8bfs3+JtQ8OWD2FlHqupwOjWcFr8/2WCTfsj4/wCWiV8/6f8A8eNp/wBcEr0P/gl7/wAmeeLP+xg1f/03Wleeaf8A8eNp/wBcErx+IKUKNSNGB6fDtWVaNScyaiiivkD6zU4/4y/8kh8a/wDYP/8Aakdee/sqfsw/Gz40/Ci81f4f/Ei18LeGV1O606fS59Zv7QPOYIPOfy4IXQh45I0yeuMY4r0H4zf8kg8bf9g//wBqR17h/wAEs5pLf9kPxXPE/lyp4i1R0kT+B/sNpX6Nw5X+r5dWqQ/nPz7iGHtMbRh/dPBNH/4JYfHPQI3TS/iL4O09ZvvpZ6vqCb/rstOavj/gmf8AtDD/AJqp4bP/AHHdU/8AkWvW7P43fEOS1gd/G2q+ZIifxR//ABupP+F1fEP/AKHbVf8AvqP/AON1EuJYSfvw/wDJSv8AV3EU1pP8TJ/Y913QNF/aZf4ezWYn8ceHtO1HS9T1KKyhKXF1AqJPIlxjzpN7xv8AfTvXW/tG/wDJaPEX+7Zf+kqV4J+xNfXWpf8ABRTxhd3k73F3MdfkeZ/vu+Wr3r9oz/ks/iL6WX/pKlY8QUY0cPD2X2zfJKs62Pmq32IHm9FFXLOzTy/tVz+7tE/8f/6Z18KfchZ2aeX9quf3don/AI//ANM6+XPDPgPxz8ZP2uvE+geAfEaeEvEEl7qn2bUGvp7NIoU8zem+BHf50yOnz96+oLy8e8k+f93Gn7tI4/4K8w/Yo/5SJ6z/ANd/EP8A6Lkr7vhefs5V5Q6RPieJofuaf+I6Vv8Agmt8edQ1y61TWPiR4W1a/uDH5moT61qL3YeNNkbrM9rv3ogC/wC5Vdv+CZf7Q0Y+X4qeG9n/AGHdU/8AkSvpf44fE7xj4f8AirrOmaT4mv8ATNOgW1EdrAyeWm+FHf8Agri7P43+PI5Nk3i/VZIJP9qPzE/8h111eI4wqck4Hm4fIasqaqU57nzv4++F+q/sd6Slt8YL/T/HE/ijzv7Il0z/AImn2XyBsnR/tUabN/nw/c/55195ftZfvPhv4UdPuPqkEn/kjJX52f8ABQDxZ4i8SReAf7b1y41yCFdQ+y3E4TPz+Rv6fRK/RL9rD/kmvhD/AK/4P/SGSnmCp1sAsbT+3/7aVg/aU8xhg6n2D5cooor81dz9GCvmj9tL/V+DPpefyhr6Xr5q/bS/1Pgr6Xv8oa+l4b/5GdH5/wDpJ87xB/yLav8AX2j23Xv+Cb/7RnjrRrW2174teHtd01dlzFa6t4g1SdE+T5HCSWvHycVBbf8ABMr9oCzhSCH4o+F7eCFNqJHrmpoiJ6f8enAr7j/aI8Vaz4O+HfhS70PVrjSbme6ghea1GXdPsMkmz/xyvn3/AIXV8Qh08bar/wB9R/8Axuvq8Vn/ANXn7OcD5TA5JVxFD29OZ47qP7E/xX+COm3nj/xz440LxD4O0FPtup6Taane3T3MHCbEhngSN/vpw7ivqX9iXxboHjv9nPxhqHhyxbTLJNY1KB0a0gtt7jT4Dv2R8fxpXgvxy+KnjXWvgr47tNT8U6lqFnNpjxyQTsmx/wB5H/sV3v8AwSxU/wDDH/jf1/4SPVP/AE22lOlWpZlhp4yCtOPu/wBIivSr5fOGGqz+P3jgdP8A+PG0/wCuCVNUOn/8eNp/1wSpq/Lne5+pLZBXHfGb/kj/AI2/7B//ALUjrsa474zf8kg8bf8AYP8A/akdduX/AO90/wDHE48X/us/8Ejzv9lT9l/41/G74T6rq/w8+I9r4U8NJqc+m3WmXOs39oJp/Jg3v5cEbo+9J0Tn6V6L4X/4Ja/GrwwztbePvCNtbSYfybTWtQhSV/4N+y1r3D/gle0+m/sh+Krsbo5F8S6nPD/wDTrT5/8AvtKzbf43/EO4t45n8barvdEk+9H/APG6/U80zhYGfs5LQ/Lcsyqpjo81OZ5Zdf8ABNv9om6keWb4seGDI/f+3dUz/wCkld5+yB4t0Pw9+0zbfDe7t/tHjHRLbUdL1PULeyhENxdWsJSeZJ8CaTzPLf53Tfz9a2/+F1fEL/odtV/76j/+N14d+xrqFzqX/BSDX7q9mae8muvELyzP/G5SWuLC4yjm/OuT+H7x34rC18qh70+fn90+kP2kbN4/jJr0/wDyzdLXZJ/26pXl1eqftA3n2f40eJUdPMgdbLfH/wBuqV5peWf2fy3R/Mgf7klfnGK/jzPv8B/utP8AwFaiiiuNXudjvY+XvB/w18b/ABU/a28T+HvAHiRPCfiafUNUlh1KS8ntQkaGR5E8yBHf7g9K9w1b/gmF8fdY8Rf8JBf/ABL8K3mufJ/xMp9Y1N7v5E2J8/2Tf9zH4VkfsU/8pGta/wCu2v8A/ouevrf45fE3xh4f+K+tabpXiW+0zToFtfLtYGj2JvgR3/5Z/wB+v2PG5p/Z9CldfZPyXD4B47FTpo+Yx/wTP/aG24/4Wr4b/wDB7qn/AMi1z/xA+Fur/sd6VDH8YtT07xzN4oE6aRNpm/VPs32VB54cXscezebiD7n/ADzr3/8A4XX8Q/8AodtV/wC+o/8A43Xyx+3p421/xlpvgEa7rF3q5tpNQMDXYTMe8Wu8cf7leZg81pZrXWCnC3N2PUr5disrp/XIz+E/Rj9rOTzPh34UkT7j6nA//kjJXy5X0/8AtXf8kz8If9f8H/pDJXzBXxebq2JsfW5HrgosKKKK8PU9o+bP21v+PLwT/v3v/tCvbPEH/BN/9ozx1o9tba98WvD+uaapS5htNW8QanOiMU+VwklqcfJxXif7aX/Hn4J/377/ANoV+pH7RnizWPB/w88L3eh6tc6TdT3cMMk1qMu6CxkfZ/45X6lgMbLAZRQnb+b/ANKPzLMcN9azadFf3f8A0k+GrX/gmT+0BZ2qW1v8UPC9vBCm1Ej1nU0jRP8AwE4FSy/sM/GD4P2F5428ZeOfD/iPwt4dhfUdT0q31S9u5Lm1j/1iJDPAkcnT+NwnNeyW/wAXviNeT7IfGWq7/wDej/8Ajdcv8bfi54u1T4O+NtLuPFGpajZzaNMk/niP996/wfcxXPSz+jiWqU4fHodTyPEYde0jP4D239jDxT4a8afs8+L7vwlZvZ6dDqmpWs3+hw2vnTCygJk8uP2kSvnvT/8AjxtP+uCV3v8AwSz/AOTPvG3/AGMmr/8AprtK4LT/APkHWn/XCOvE4jowoVIUYHp8O1ZVlUnI1re4S4g+y3L/ALv/AJYXH9z/AOwqpcW72c+yb/WVFWhb3CXEH2W5f93/AMsLj+5/9hXyfxn2Gpw/xT/5JT40/wCwTNXmP7Kf7L/xq+Nvwn1XVfh58RrXwp4aj1W40260261q9tBPP5EG9/LgjdH3pIif8AxXqvxgt3s/hf41jmT95/ZM1euf8ErZ5LX9kjxlLE3lyx+JNTkR/wC466dY81+icPVJ4fAVqn9//I/PuIYe0xtGH908H0f/AIJX/HTQfO/sv4heDtOEv3/seragm/8A74tOa0R/wTP/AGhR/wA1V8Mn/uO6p/8AItetWfxu+IUlrA7+NtY8x0T+KP8A+N1J/wALr+Iez/kdtV/76j/+N1nLiWEn78P/ACUpcO4mmtJ/ic/+yTr2i+Fv2nrP4c39klx460Kz1TS9S1O3s4ylzdQQnz5Euv8AXSb/ACz/AKyOu3/aM/5LR4i/3bL/ANJUr58/ZFvrnVf+ClGv3t5M93dzTa87TP8AfdvIkr6A/aIbd8ZPEJPpZ/8ApMlYcQUY0cND2P2veNslrTq4+SrfZXKec0UUV8HqfdBXzN4X+Gvjb4qftgeKfD/gDxIvhPxLNfalJFqMt7PahY03vInmQI7/AHBX0zXn/wCxr/yki1r/AHtc/wDRMlfb8Lz9nVrvtE+N4m/gwX94vat/wTB+PeueITrmo/Evwjfa4dn/ABMbrWtRe7+RNqfP9l3/AHAPyqUf8Ez/ANob/oqvhv8A8Huqf/ItfTXxw+KHjDw/8VdZ0zSvE19punQLbeXawNH5ab4Ed/4P79cJ/wALq+IX/Q7ap/31H/8AG69GrxNCM/ZzgeZh8hq1aftKc9zwH4gfCvVv2PdJij+MOpad44m8TiZNJm03dqn2b7MgE4cXsabN5uIPuf8APOvu/wDaybd8NfCki/ck1SCT/wAkZK/Of9vPxvr/AIy03wENd1i71Y20moGBrsJlN4td4490Ffo7+1JaPefDXwhGh8tEvYJHeT+D/QZKrGyhicuWNh9v/wBtFhfbUMxhg6n2D5gs7N7yTYn7uNP3jySfwU+8vE8v7Lbfu7RP/H/+mlF5eJ5f2W2/d2if+P8A/TSqdfnB+jaixyPbyRuj+XIn/LSvAP207UavpfhG5twkdxCdQkmhzwyfuPnT2r36vm39syR7e28FSIzI/mX3zp/2wr6Hhz/kZ0fn/wCknz3EH/Ivn/X2j2vxB/wTj/aM8daPa2Wv/Fzw9rmmjy7uK01PxBqk6I/l/I4R7Xh9j/Xmq1r/AMEyf2gLO3htrf4oeF7eCFNqRx6zqiRon0+ycCvu/wDaD8Sa74d+HehXnh/U7nSb2S6tknktNnmOn2WR8f8AoFfOX/C6viH/ANDtqv8A31H/APG6+sxWf/V6ns5wPkMDkdXEU/b05nkFz+w/8Xvg9Y3fjnxn458P+JPCXh6Br/VdKt9SvLmS5tYzmSNIJ4ESTp9xyEr6m/Ya8XaB46+APinUfDtg1hYprWoW7o1nBbb3+wwHfsj4/wCWkdeG/GH4reNdZ+Dvjqz1DxXqV9aTaNMj29wybJI/++K7X/glX837I/i4/wDU0an/AOm6xp0q1LM8PUxSVpR93y+4VehWy+pDDVZ35jz3T/8AkHWn/XCOpqh0/wD5B1p/1wjqavy53ufqa2QVzPxQ/wCSWeMv+wVN/Oumrmfih/ySzxl/2Cpv5124P/eKf+OJxYz/AHeoeYfso/sx/Gr41fCnUtZ+HXxHtvCPh2PV7jTLjTbjWdQs/On8iF3fy4I3Q70kjTJ67MdhXcaP/wAEr/jn4f8AO/sv4heD9P8AN+/9j1fUE3/98Wle9f8ABKeR4/2TfFcifu5E8U6i6P8A9w+xqhZ/G74hSWMDv411jzHRP4o//jdfqmZ5wsBU5JLQ/L8tymWOjz05nkv/AA7P/aF8rH/C1fDH1/t3VP8A5Fr0D9krXNF8K/tPWnw41CyS78c6Daappep6tb2abLm6ghPnSJdf66Tf5Z/1kdb/APwuz4h7P+R21X/vqP8A+N14p+yJqF1q3/BSjXry7me7u5Z9ed53++7+TJXFhMbRzeT9z4FzdjuxeFxWVQ96ek/dPoT9pT/ktGtf9e1l/wCiUrzOvTP2lP8AktGtf9e1l/6JSvM6/N8d/vUz9CwH+6U/8AVcs7NPIkurn93aJ/4//wBM6LOzTy/tVz+7tE/8f/6Z0y8vHvJN7/u40/dpHH/BXMjq6HzPo/gPxz8ZP2vvE+geAvESeEvEEl1qLQahJeT2SRQoj703wI7/ADoCOnz9+te23H/BNX48avrkmo678RvCur6gxT/iYXetag12rohSORJntN/yVifsct/xsg1ge+uf+iZa+r/jx8TvGHhv4panpukeJr/TNOhtrV0tYGj2Jvh+f/lnX67i8z+oYWlK32Yn5RRy147F1qdM+ZG/4Jm/tDRLhfip4bKf9h3VP/kSsLxr8MdX/YztLP8A4XLqeneOD4ld/wCyZNJLap9m+zf8fG/7bHHsDmeH7n39lfQFn8cPHscmybxhq0kEn+3H5if+Q6+WP2/vFXiPxNY+ARruuXGuQwyah9knuNuV3i13jj6CvPweaUc2qrBThbm+R6FfLsXlcPrPP8B2Xjb4mftk/ESztLbWvhZrcsVnO88Zi8FOnz7Nn/POtW18daPplna2viTX9K0nxFDCianY3VykE9tdbMTo6fwOkm/5K+8vjp8UtS+FGj6Pd6bp9lfveXLwPHfb/LTYhf8Ag+lfPuq/8E4fhb8Y7+5+IOseIPFtjrPiuR9fvrXT76z8iKe6/fuieZBv2B5MDfU4ingMzl7OsuSZnha+My2HPT9+EzzCwv7bVrGC9sbmG/tZvuTwPvR/+Wf36q+IL6bSfDOt39uUF1Z6fdXUPmR+Z86QSOn/AKLrzH44+LtR/Zf+KWtfCzwhoy+IvDHhnyIbLUtUjmkuyJo47pw7wmND887/AMFc14D+PGu/E/4i+Gfh/rfh6x0qw8V6ha6JdXFrFPHdwQXUnkO8O+TZv2SPs8xH615EeGsUqyqQSdP9D6B8QYb2Vp/xD3X9iv4W+F/21vDfijUfippj6vJoN7a2WnnT52sBDHMkjycR/f8AnRK7/wAZfFDxV8WVtbbWp7O4S2neeH7LbJD8/wBz5/8AgFdl4e+GemfsIyJoXw7F74ih8TNJeXTeJ508yFrXCJ5fkxp189/v+leT3FwlnBJa2f8Aq3+/cf3/AP7Css7xceZYeh8Bz5FhZuTxdb4pbElxcJbwSWts/wC7/juP7/8A9hXnfxs5+DnjX/sHj/0ojrtK4r42/wDJG/G3/YP/APbiOvncF/vVH/HH/wBKPp8V/utb/BI8+/Zv+Jf7SOgfBm98O/DHwZe+IPBl1d3SyXVp4b+3fvpI40nTzAP7myu78G6x408L6Tdf8Lg0xvAt3LLGmlR67ZJpf2lEXE+zeBv2fuv+/lfQH/BNPUn0X9i/WtSgjSSez1nVrlI5PuO6Wtu//slamrfB7QP2+7O1v/iBc33huXwu/wBmsV8NTxxpMlygkcv50b/3K/TczlhMXXeFrq0v5j80y2pisJS+s0Ph/lPE9J8SaP4gknj0nWLDVpIY/MeOxuUm2f8AfFadZH7RnwC8PfsK+GtJ1z4dS6v4mvPEV6+nXdv4hkS4jREQTJs+zJH8+8evavnmb9rDxhZw+ddeENLtovueZJBeIPz82vlqnDVepUvg3z0/M+qocQYbktivcmezeCfEUnxW/au034JeKIIrzwJqc5guI44/Iutn2XzxH56Dfs8+NK+kfFFvF+y60vw4+HdvBpfhW8s/7Rngvt97J59z5iT7J3/2IErhvgP8F/Dz/D/w/wDtWLcainj+KzuNSXRxNH/ZW6OSSy2GPy/O27E/56ff74rR8cfECb4ma4t7q9lbWE62yWySWu/YmzzP7/8A10rtzPEU8DhaeEpe5PTmODL6M8fi54mfvw+ycXHH5ccaJ/BTqluLd7OfZN/rKir8/V7n3etj5m+GfjL4o+C/2ofH198JNEn1/wATPPqcMlrBph1B/sr3Pzv5Y99nPvXrDeKP2h9b8dnxH8U/BGoeHPDc8kcmratd+GvsNpDGkexJHn8v5P4K0/8Agn23/GfHxEAH/Lhrn/pSlfXvxa1o/F3xtrPwP1m2Sz8L615Njc6jYuI76NBAl1lN/wAn340T7lfsWOq4ZUoUa8L8/Kj8owkcSsVOtQl8B8qQePvCt3MkUXijRJ5pn2IiX8Lu7+tbldn4g/4Jn/Cv4b+GNY8WaV4g8X3uq6DYT6xaWtxfWTRyTwQPMiP5cG/YXjx8lfGS/tVeNpmL/wDCGaXK7947S9/+OV8riOHfa/7g+b1Pq6HEEf8AmL9w9P8Aj98RdZ+GfhPStT0OSGKe5vZIH+0QJP8AJs3/AMYr6Y8H/C3w18HvhF4W+OHhWwktfiJ4n0fT59T1C4uJJ4Jn1CCOa6/cfcT56+df2Y/DOl/t7al4g8N+N1uPD+neHbVNUt5PDDCORndxC+8z+Z8myvoPxv46utN8Jr8KorO3Hh/ws0GjWV7JI5upoLLECO/Gze6R/PsTvXTUislwHspe5W8upwRqPNsw5oe/Rgcf4k8SX/jDXbvWdTeGTUbnZ58kEOxPkTZ9z/cSsmirNnZveSbE/dxp+8eST+CvgL+0qXZ92kqVO0dj5p/a+0xtTvfBsaOqKttdPNM33ETenz17D8Svil+2N8Q4Lazv/hdrstpbzeenl+CpEkd9mze/D14/+27cow8E21tujtFhujyeZP3ifO9frF+0H8TtS+FGl6ff6bYWd/Jeag9o6X2/y0/dySfwV+oYWtRw+UUPbQ54e9/6UfmWOpVq2bT9hLln7v8A6SfAdr460bS7S1tfEmv6VpPiKGFE1OxurlIJ7a62YnR0/gdJN/yVu2N9Z6tZpeWNzDf2s33J7V96P/yz+/Xq2qf8E3/hX8ZtQuviBrPiDxZZa14qc69fWthe2aQRT3X7+RE8yDfsDyfx18s/HTxJqH7K3xS1j4V+D9EXX/DHh1IPseparHNPdyCeFLpw7wmOM/PO6fcrxsRkVHFRvgZ80/wPXwmezhLkxkOVHpmtXj6foOs3UWzz7Wyup08xN/zpBI6Uv7Ffwr8L/tqeHPE2o/FPTJNWn0G9tbXT30+d9P8AJSZJHc4j+/8AOifnXhPgf9oLxD8QPiB4c8EatoGnaRaeJb2DR7qe3hnSdILp/IeRN8n39kj7K+5NH+H+l/sDj+xPAP23xBa+JpZLy6PiadHkhe1+RPL8mNP+en8ddeDwv9iUZ1MVbn+yc+OxTzapCjhZ+4cf40+MXin4iQWkOs3VnIlrcvdQfZbNIZEcfJ3rnZI01SOSeFPLnT79vH/H/wBNI6z6WOR7eSN0fy5E/wCWlfBVMROq71D7mlh6dFezpnF/Gr/kjfjT/rwH/o+OvP8A9mf4pftJ+Fvg/daD8LvBl7r/AIRnvbnzLqDw39u/fSRxpOnmY9PLr1L44RpqnwW8azwxiOePTxJPbr0/4+I/3iV7d/wTLvG0z9jjU75ESWS11rVp0jk+4+yCB/8A2Sv0Hh6osPl1ac1f3z4DiCHtMdTh/dPnfwbrHjTwvpV1/wALg09vAt3MyJpMevWSaX9qRFxPs3gb9mY/+/ldTpfibRdeuHg0rV9N1WSFPMeOyuUm2R/8Ar2vVvg7oH7fdhaX/wAQLm+8Ny+F3+zWK+Gp440mS5QSPv8AOjf+5Xin7RH7Ovh79hnwnpniT4eXGr+Jb/XNQ/sqe18QSR3CJEiefvT7Mkbht8Y5zXHUyrB5hB18M7VJ/Z6HdSzbFYGfsMSvg+0aVefeCtWm+KH7XWl/BnxEsN14G1ZkivLWGPyZpE+xC6A85Pn/ANZGnevI5v2sPGFnH5114Q0q3g+55jwXifr5tfXX7Pvwb8O6h8OfD37WHm36+P4rW7vBoyzRjSt0E0mnp+72ed9yPf8A6z7/AOVaZflE8qnPEYyK5OX/AMnIzHNoY+nChg5+/wAx3viWCP8AZfaT4d/D23g0vwre2f8AaU8F8HvZ/PufMjn/AHz/AOxAleRxx+XHsT+Cup+Injy9+JHiJNWvrK2tJ0tUtdlrv2bEkk/v/wDXSuYr47HYqeMrc/OfUZfhYYShyOGoV8wfD/xh8TfBf7V3jTUvhRos2u+KvtmpxfZYdM+3P5DSv5h8v8q+n64D9hH/AJSHeLf+uWu/+hPX1nCk/Zuu/wC6j5vib+DT/wAQjeLP2h9c8dHxJ8U/A+peHfDc8kcmr6vdeG/sNvCiR7Ed5vL+T+Cut0vxp4QvNrv4p0eRGfYscGoQeZM/9xK+q/ireSfF3xdrvwX12KGx8J62YbKbVbCTy77/AFCXXyb96ffT+59yvI/F3/BN/wCGPgnQtU8Y6d4h8YXWqeHNOuNUsLN7qy8kyW0ck6JsSDfsLpj+/XZWweBzOcpr3JrsceHzHG5dCFKfvQ/9tOMvLx7yb5/3cafu0j/uV5V8fviNrPwy8J6VqehyQxT3N7JA/wBogSf5Nm/+MV5iv7VHjadjt8GaXKzf3LS8/wDjlewfsx+F9L/b41DxB4b8cJPoGn+H7VNTt5PDLeXI7u4h+fz/ADPk2VyYLIMTha8a+KinBHo4zPMNVoTp4Wfvn0R4P+Efhb4TfCPwv8c/DemG1+I2v6Rp13eX89w89q8moQRvd/6N9xB871xHibxNqHjLXrvWtUeGTUrrZ58kEOxPkTZ9z/cSuv8AHHj670/wifhRFZWy6B4ZaHR7K8eRzdTQ2BEMDv8Awb3SP5/kx+lec14mc476xU5IT9w9HJcH7Gj7WcPfCvnj9rzUo7ObwbFdw+fZTQ3QaP8AjT94nzp/t19D18zftrf8yV/1xuv/AENKrhv/AJGdG/n/AOkhxB/yL5/19o9r+Inj/wDbH8dQ2ljqvw11e7tLWT7TC8Hgspzs2eh7VYtfHWj2Fna2viTX9H0nxFDCianY3VykE9tc7MTo6fwOkm/5K+/fjp8UL/4U6To1/YWFnf8A2y5+yvHdb/kj8gv8myvnjWP+Cbvwq+MOpXfxA1bxH4tttW8VTSa9dwWF3ZJAs9z+/dE8yDfsDvgb+1fW4mjgczn7Ot7k4HyeBxGMy2ipw9+EzyaxvrLVrNLyxuYb+1m+5PavvR/+Wf36j1q7ew0HWL2LZ59rp11OnmJv+dIJJErzP47eJNR/ZX+KWs/Cnwfoi6/4Z8OpB9k1HVY5pruTz4UunDvCY0Pzzun3K5bwT+0Br/xC8e+G/BWraBYaRY+Jb2DRria3juUuI4Lp/IeSMPLs37JH2HFeD/q1ilW9pTd6f6H0X+sGG9naf8Q92/Yp+Ffhb9tjw74o1D4qabLq82g3trZafJp872XkxzJI7nEf3/nRPzrvvHXxY8R/EaCG112azktrO6eeGOC2MD7/AN4n3/8AcrrtJ+Huk/sBs2j+APtuu2niSV726PiadHkha1cInl+RGnXz/wCOvIf+Wkj/AN95JKWfYuOmFw3wGORYWcpTxdf4pbDa4v42f8kZ8a/9g8f+j467SuL+Nn/JGfGv/YP/APa8dfM4H/eqH+KP/pR9JjP92rf4JHnX7MvxM/aO8M/CHUdC+Ffg298QeDrrUblri4s/Dh1H/SHgjSRN4H9wR133g3WPGnhfSrpPjBp7eBbuZkTSY9esk0v7SiLifZvA37P3f/fyvef+CXd8+l/se+Jb2ONXktvEepzJHJ9x9ljaP/7JW3q3wd0D9vvT7S/+IFzfeG5fC7/ZrJfDU8caTJcoJHL+dG/9yv1DNXhcVW+qV1b+8fmmW1MThKX1mh8P8p4ppXifRNfknj0rWNN1WSFN7x2Vyk2yP/gFaVZf7RH7Onhz9hnwnpviP4d3GreJr/xBf/2VcWviCRLhEhRPP3p9mSNw++Mc5rwaP9qbxXpti2pan4L0uO2z+5WS3vYy4/3/ADPufjXy9XhqvOd8G+en5n1VDiCjyf7V7kz13wPJJ8S/2qNK+EnihIZvAutyJbXFkqiC6uU+xfav9cn7xE85I/8A0D+/X0j4qVP2Y5Jvh/8AD+GOw8L6pZ/2jdQak8l6/nz74X2O/wBxNkEfyV598B/hPoOsfD/w/wDtaSXN/H8Q1t7q8XR1mjGkboJpNPjTy/L87aY4x/y0+/3xU/xE8f3nxI1+LV76ztrC4itksilrv2bEd3/j/wB+ujNcRDA4WGEoe5P7X+I4Mtozx+Lnia3vw+yctHH5ccaJ/q0p1FFfB6n3mtj5l8B+LviZ4K/au8Zah8J9Gn13xV9q1OP7LDppvn8h5D5h8sfhXrJ8TftE6946XxH8UfAuqaB4dmmjk1fVrzwubKCGFECb5JPL+ThEqx+w3/ykM8Wf9cte/rX2R8VPE0nxO8daz8FdXtobfw1r+zTbrUbRxHfRo8CT5Tf8n3/9iv2LHVMMqFKjXhfnjE/J8NHErF1q1CfwHy5o/wATPC0kq26eJ9FvFuGjU2sd/DvZumU/266C80/7H86P5kD/AHJK6fXP+CZvwr+G3hzVfFuleIPGF7quh2M+sWkFxfWbxyTwQPMkb+XBv+/Hj5M18e2P7W3jbcY5PBOnXMM/3o0t7395/ufvK+WxHD3tV/wnvn9T6ehxBHbF+4ei/tAfEXWPhn4U0rU9Ekhinub2SB/PgSdCmzf/ABg19MeD/hF4V+E/wh8KfHPw5p7QfEfxBpGnXd7qE1y88Dyagkb3f+jfcQfvJP8AcrwD9nPwXpP7dGseIfDXjKd9H0jQ7FNYs7jwu4SdneTyNkxn8z+D+Cve/HPjy60nwqfhLbWlsPD3hc2ujWV47t9qkjssJHJJxs3v5fz/ACd67KkVk2A9lP3KxyRqPNsdzQ9+jA4vxN4l1Dxhr13rOpvDJqVzs8+SCHYnyJs+5/uJWTRRX59Uqe0ftGfd06apr2aPmj9tT/j38GfW9/8AaNev+OPiR+2b8QrK0ttY+FutyRW0/np5Pgsp8+zZ/wA868g/bT/49/BZ97z/ANoV+tPx3+JepfC3Q9Jv9NsLO+kvLv7K8d9v8tP3Jk/gr9PwVajQyih7aHPD3v8A0o/NsdSrVs2n7GXLP3f/AEk+B7Px1pOl2Nta+JvEGlaV4lghRNT066uUgntrnOJ0dMfI6Sb/AJK3bG/s9Ws0vLG5hv7Wb7k9q+9H/wCWf369Ru/+Ccvwu+Nl5P8AELW/EPiqx1rxW51++tdPvbJIIp7r9+6J5kG/YHk/jr5e+OviTUP2V/ilrPwp8IaIuv8Ahjw6kBs9R1iKae7kE8KXTh3hMcZ+ed0+5Xk18io4r95gZ80/wPXwuezhLkxkOVHpmtXj6foOs3sWzz7XTrqdPMTf86QSSJUH7EXw18Nftuad4tn+KenNq7eHbqxh099OuH0/yluUn84vs+//AKiOvEfBX7QOv/Ebx14c8E6toFhpNj4lvYNGu5reOZJ0gun8h5E3ybN+yR9hr7n8P/CzS/2DLmTTPh2174ij8Vyma7bxLcI/k/Y/kTy/IjTH/Hw+/f8A88668Hg/7EpTq4mP7z7JwZjinm1SFHCz9w5Dxj8UPE3xNWHTNXezuILO6eeGOC2SHZ/rE+d/9yuZuLhLeCS1tn/d/wAdx/f/APsKLy8S3jkhtv8AVu/mTyf3/wD7Cs+vz+pVnVd6h95SpU6MbUzj/jJ/ySHxr/2Dx/6MjrzX9l/4nftG+GPhDqOi/CzwXe6/4PutRuTPPbeHP7R/fvBGkieZj/nnsr0n4y/8kg8bf9g//wBqR17f/wAEur99I/ZA8S3yokj23iLVJo45P49ljaP/AOyV99w9U9ll1ac1f3z4TiGPtMbRhD+U8D8Fax428L6fdf8AC4tMfwDPM8aaTHr1l/Zf2pER/P8AL3gb9mYP+/ldZpfifRNelki0rV7HVZYU8x47K5SbZH/wCva9Y+Eug/t+6fazfEO4vvD3/CKMVsf+EYnSPzvtSB33+dG//PFPuV4p+0R+zx4d/YZ8J6b4h+Hlxq/ia/1+/wD7KntfEEkdwixInn70+zJG4bfGOc1z1MsweYQdfDaVJ/Z6HVSzbFYGfsMSvg+0aVcD4V8WXPxH/aq0j4LeIY4bvwPq1zDbXCLHsujC9r5/l+enz/fryGX9q7xhaxvLP4Q0uKBfvyPBeJ+vm19a/AP4I+Hta+G3h79qqS71JPH0UNxqa6NFNGNL3Ws8llGnl+WZtvlx/wDPT7/fFbZflE8rnOvjIrk5f/JiMxzSGPpwoYOfv8x6D4k0+3/ZVWT4e/Di1TS/CepWf9q3VvfeZeu88++CTZO/3PkgjryOOPy440T/AFaV2vjn4iTfE3WIr3V7OzsJIrZLVJLXf+72O7/Pv/365G4t3s59k3+sr4/McVPGVufnPqMtwscJQ5HDUir5j8D+LPiX4M/ay8Z6j8KdFm13xX9s1SM2sGmfbn8h5D5h8uvpyuA/YZ/5SH+Lf+uevf8As9fUcLy9nKu/7qPn+J/4MP8AEI3ib9ovxB46XxF8TvAuqaB4dkeOTVtWuvDX2G3hhRAnmSSeX8n3Ero7fx/4TvJkhh8U6HcTzPsSOPUIZJHevq74teJpfiR461n4KatbQ2/hjXtmnXWpWsnl3yI8CT/Jv+T7/wDsV5Drn/BM/wCFnw48Pap4s0rxD4wu9V0Own1i0t57uzeOWeCJ5kR/Lg37C8YHyV2VsHgMzlOa9ya7HnUMdjcthCFT3of+2nGV5p+0D8RdZ+GfhTStT0OSGOe6vZIH8+BJ0KbN/wDGK8wX9qjxvcMXXwZpsrN3S1vP/jlev/sxeFdM/b41DxB4b8brPoGn+H7VNTt38MsI5HmdxD8/n+Z8myuHA8PYjC14VcVFOmj0sbnmGqUJ08LP3z6J8F/CHwr8I/hD4V+OvhywaD4j+INJ067vdQmuXngeTUEje7/0b7iD95J7JXD+KvFWoeNteu9a1d4ZNSutm+SCHYnyJsT5P+AV2nj7x3daR4UPwltrS2Hh7ws1ro1leu7fapI7LCRyScbN7+X8/wAnevNa8fOcd9YqezhP3D0clwao0fazh74V87ftgaY+qSeDokdUVVvZJpm+4ifuPnr6Ss7N7yTYn7uNP3jySfwV83ftvXCNa+C7a1DR2qm9xzzJ/qPnetuG9Mzo/wBfZDP/APkX1v6+0eu/ED4ofti/ECzg0/U/hdrb2NvP9oTZ4IMbvJsePe/B7OalsvHWk6XY21r4m8QaVpPiWCFE1PTrq5SCe2uc4nR0x8jpJv8Akr79+PHxJ1H4V+H9N1LTbCzv57q9S1eO/wB/lonkSP8Awf7lfON5/wAE5Phf8bbqf4h63r/iyx1nxW516+tdPvbJIIp7r9+6J5kG/YHk/jr6rExwGZT9lV9yR8ngcRjMto88PfhM8tsdSs9Ws0vLG8hv7VvuTWr70f8A4HT7+4ez0vUp4v8AXw2U86eZ/fSOR468q+O3iS//AGVfipq/wt8G6F/bvhjQFtZLbUdYjmnu5RPDHdOHeExp9+d0+5XIeGf2jvEPi/xlofhLVfDum6XB4guoNMnnjjuUnSGeTyJHTfL9/DvzXz/+rOL9rem70/0PoocQYT2dp/xD3X9iT4b+Gf21tN8X3HxT0w6yfDlzZQ6e+nTtp/lLcpP52/yQN/8AqI+teleNvi/4p8fRw2mr3Vn5Nne+fD9ktjA+9PMT7/8AuV0+i/DPR/8Agn+82n+AZL/xBb+KJfOvD4mnjcwfYn2J5fkRp9/z3+//AM868k/1kkj/AN95JKM+xcef6rhvggY5FhZVo1MXifilsX5I01SOSeFPLnT79vH/AB/9NI68/wDjJ/ySHxr/ANg8f+jI67GOR7eSN0fy5E/5aVg/GiNNU+D3jSeFFjnTTHknt16f6yP94lfO4H38XT/xxPo8X7mFqf4DyP8AZh+J37R/hH4U32j/AAo8GXniDwjPqVzJNd23hs6j/pTwRpIm8D/nmI/zruvBOseN/C+m3Q+MemN4Bld4E0mPXbL+y/tUaI/n+XvA37P3H/fyvoD/AIJZX32H9kfxDdIiPJB4m1KZEk/2LG0atfVPhNoX/BQDS7Gf4hz33h7/AIRUbbFPDE6RiX7Ugd9/nRv/AM8U+5X6XmjwuKrfVMRD/t8/NctlicJH61R+H+U8V0rxPoniCeSHStb03Vp0Qu8djcpNIif3/krSqj+0L+zP4W/Yd8D2HjD4e3eteJdT1rUU0O4sPEDxz2/k7JJ96fZkjcPvgj7/AN+vnCX9rDxhawvLP4P0uKJfvyPBeIPz82vmKnDVSpO+DfPDzPqqPEFHk/2r3JnrXh7XpfiF+1to3wd1uOKfwTrU1rbX1tGmyZ0e1Sdx53+s/wBZ719O65Z2/wCyqzeAPhlAml+F9Rt/7VurfUvMvZJJ5t8L7Hf7nyQR1wHwD+C/h7XPhv4d/arml1JfH1vDcahHo8U0Y0rdazSWUaeX5Zm5jjH/AC0+/Vz4ifEC9+JHiCLV7+ztrC4itksjHa7/AC9iO7+Z8/8Av10ZnXp4HDQwtD3J/a/xHHl9GeYYueJn78PsnLRx/Z440T/Vp+7p1FFfCXdz7l3sfMvgXxd8TPBv7V3jHUfhRo0+u+KTeapH9lh0z7c/kPKfMPl16y3ij9orxB8QF8R/FHwRqmgeH3eP+19Wu/Df2GCGBE2b5JvL/d/wVY/YaOf+ChnizjpFr38zX2l8U9Wl+JnjDWvg7rNtDZ+FdeWHTZtRtHEd9seFJ3Kb/k+//sV+w42pQ9hGjXh8cYn5Ph411i61WhP4D5U0vxv4QvNrv4s0OSNn2rHHqEPmTP8A3ErbvLx7if508vZ+7jj/ALldz4j/AOCZ3wv8N6Ze+JLPxN4wuLzR7Ke9s7WS6sfI3wpJMkexIPub0r4mX9qrxtdSl/8AhDNLleb5/wB3aXv/AMcr5Wvw97Vf7A+b1Pp8LxBH/mN9w9S+PXj/AFj4aeDLDV9Ekhiupb/7K/nwpOhTY7j74P8Acr6T8F/Bvwl8L/hD4U+PPh7TGh+JOsaPp2p3WoT3Dz2rz6gkf2r/AEb7gH7yTj+Dj0r5u/Zn0XTv27/EGt+DvHEM+gabotj/AGzDJ4ccRzvN58cGx/P8z5Nk/wClfSPjrx/daL4Tk+EltZ2z+HvDQtdDtbx3b7U8Fl5aRyScbN7+X8+E711OKyXAeyn7lby6nI6n9s47mh79GBw/inxVqHjTXrvWdXeF9Rutm+SCHYnyJsT5P+AVk0UV8BUqe0ftGfdU6apL2dM+ev2utRjtV8GxXkXn2Uy3oaPA3JnyPnT/AG69W8efEL9sTxtp9hYah8NdXu9OgkS6tng8GFCTsMab+Dj5K8a/bRP+j+Cvpen/ANE1+s3xe+Jl98K/B/hu/sNPtb/7bPDavHdb/wDV/ZS/ybP+udfp+CrUaGS0PbQ54e9/6UfmeOpVq2bz9jL3/d/9JPgez8daRpdjbWvibxBpOleJYIUTU9OurlIJ7a5zidHTHyOkm/5K3bHUrPVrNLyyvIb62m4Sa1fej/8AA69U1L/gnL8K/jZqFx8QtY8R+LLXVvFkh1+7g0+6skginuv37onmQbwg8w/fr5a+OviK+/ZU+KmsfCzwboX9u+GNAS1NtqOsRzTXconhjunDvCY4/vzun3K8rEZFRxX7zAz5p/gevhM9qU5cmMhyo9U1C4ez0vUp4f8AXw2M86eZ/wA9EjkeOqH7Efw38N/traf4tl+KenHVm8N3NjBp76dcNYeUtyk/nF9n3/8AUR14X4Z/aO8QeLvGWh+FdV8O6bpcPiC6g0yaeOO5SdIZ5PIkdN8v38SPzX3Rovwz0n9gCa4svAT32vw+KpfOvP8AhJ50fyfsT7E8vyI0Hz/aH+//ALFb4PB/2HSqVsVFc/2Dnx2K/tacKOFkcl4++KHiDxtbnRtXms5NO0+9k+yxwWxR/k8yCP5/4/krjaluJPtE88//AD2eST/vuTzKir4OpUnVqe0qH3NKnTo0/Z0zkPjH/wAkh8a/9gwf+jI68w/ZV+KX7Rfg/wCFOq6R8JvCN5r/AIVutUne5uLTw4dR2XTwQJIm/HyfuxH+den/ABk/5JD41/7Bg/8ARkde1/8ABKS5/s/9kvxZcrGkkkHijUZ0ST/Y06xevvuG6nssurVLfbPhOJIe0xtGH908E8E6x428L6bdD4xaa/gGV3gj0mPXrL+y/tUaI/n+XvHz7P3H/fyut0rxNoviCZ4dK1vTtRnhTe8dlcpM6J/f+Sva9U+Eug/8FANKsZ/iHcX3h4eFMrYx+GJ0jE32qON33+dG/wDzxT7nrXjX7Q37Nfhf9h7wPYeLvh/d614l1PWdRTQ7ix8QPHPB5Ox596fZkjcPvgj79nrCeWYPMIuvhtKk/s9Dpp5pisDL6riV8H2i5XFeF7yXx1+1Jo3wo8RrDP4H1u5tbaexVRHdXKPaidsTJiRE8z0P+x615PH+1N4r02xbUtT8F6XHbZ/crJb3sZcf7/mfc/Gvqv4EfB7QfFHw88P/ALWF1daknxAiiuNRi0i3kjTSg9rNJZRp5ewzbNkY/wCWlaZflM8rnOvjI+5/7cTmGbU8dThRwc/fPRvFkMX7LpPgb4cxx6Z4b1W3fVbqDUhJeyefN/oz7Hf7ibII/krxmOP7PHGif6tP3ddb8RPiBe/EjXLXVL6ytrC4t7ZLZEtd+zYju/8AH/v1ylfGY7FTxdfn5z6fLMLHCUOVw1CvlvwX4s+Ivgz9rfxVqfws0afWvFa6hqaQWkOnfbn8t3kEh8v/AHM19SV5x+xR/wApE9c/67eIf/RclfWcKP2dSu/7qPneJv4NP/EWW8UftFeIvH6+I/ij4E1TQPDzvGdX1a78N/YbeGBE2b5J/L+T+Cup0v4neFCxtk8UaJeRSjaYY9Qh3u/qn+3X1P8AGDxNJ8RvHOr/AAT1e2hg8L69HDp11qNo4jvkR4En+Tf8n3/9ivJdU/4Jf/CjwVo1/wCJNO8SeNLi+0m1n1GCCa+snjd4Y3mRH2Qb+XSu2thsBmc5zXuTXY4MPjsbl0IQn70Dk9Q0/wCx/On7yB/uSV5b8e/iBrPwz8Gafq+hyQx3Ut/9lfz4EnQpsdx98H+5Xm1l+1x448xhJ4J029gujvaNbW8/ef7n72vVv2d/DOmftzeKtZ8GeMnk0jR9J0865b3Hhh/LnaTz44Nk3n+ZzskeuHC8P4jC14YjExTgviPTxWeUatCdOhP3z6H8GfBvwl8LvhD4V+PHh7TWh+JWr6Rp2pXWoTXEk8Dz6gkf2rFt9wD97Jx/B+FcP4q8Vah421671rV3hk1K62F5IIdkfyJsT5P9xK7jx18QLrRfCcvwktrO2bw94ZEGh2t47v8AangsvLSOSTjZvfy/n+SvMa8XOcd9YqezhP3D0cmwao0fazh74V82ftrf8ePgn/fvf/aFfSdfNv7aX/Hj4K/66ah/7QrThv8A5GlH5/8ApIZ5/wAiyt/X2j1fxd8Sv2yfiBpNjp2rfC/WJLS1dJ4fI8FFP+WZT7/l/wBx61tP8bWGl6bZWfinW9L0PxPDCqalpt/dJaz21yD+8R4cfJ/uV94fGL4kX/wr8E+H7/TrKzv3vXgtXjvt/lpH9l8z+D/rnXgF/wD8E8fhn8fLyT4j6/r/AIp07XPFY/tu9tNMvbJbeGef946J5kG/YM/x19ZiqWBzOfs6vuSgfJ4HEY3LaCqU/ehM8qsdSs9Ws0vLG8hv7VvuTWr70f8A4HT9QuHs9L1KaL/Xw2U86eZ/fSOR468r+O3iO/8A2VfiprHwt8GaF/bvhjQFtTbajrEc093KJ4Y7pw7wmNPvzun3K4/wz+0d4h8XeMdD8Jar4d03S7XxBdQabPPHHcrOkM8nkSOm+X7+JH5rwP8AVnF+1vTadP8AQ+ihxBhPZ+//ABD279iX4f8Ahv8Abet/GafFbT21ZfC82nppn9nyf2f5X2r7R55fywN//HrH+tereNviR4m+IQTQ9RezkstPvZHtY4LYo6bPMgj3v/uV03h34P6P+wVe3Fn8OmvvEQ8XS/6afE08b+T9iz5fl+RGn/P1Jv3/APPOvM9QvE8ydLb/AJbPJJPJ/f3yeZ/3xWmeYuD/AHGF+CBzZFhZ1ZPF1/ilsR3FwlvBJa2z/u/47j+//wDYVxPxQ5+FvjIf9Qqb+ddNXM/FD/klnjL/ALBU386+Wwf+8U/8cT6nE/7vUPKP2W/ih+0V4Q+Feq6N8JvCF74g8KXOqzvcXVn4d/tHbdvBBG8e8D5P3Yj/AO+67zwXrHjfwvY3CfGLS28AySGGPSU16y/sv7THHv8AO8veBv2fuf8Avuvff+CVNz/Z/wCyb4quUjV5IfFOoTqkn+xp1i1beqfC3Qv+CgGm2H/Cwpbzw6PCgzZHwvPGgm+2pHv8zz43+59lT7n9+v0/NPquKrfVMRC394/MsrqYnCU3iqHw/wAp4tpXibRPEEzw6Vrem6tPCm947G5SeREH8fyVpVT/AGhP2Z/C/wCw74HsfGHw9u9a8S6prWox6HcWHiB457fydkk+9PsyRuH3wR9/79fN037WHjC1heWfwhpcUS/fkeC8Qfn5tfL1uGqtSd8G+aHmfVUOIKPJ/tPuTPXdF8Z3Xjn9qTQ/gxraQ3HgjWru1sLqOOLy7jyZoI5HTz/v/fr6f8R6bb/snr/wgHw1t4dJ8N6pavqlzBfk30j3U2+B3SR/+mcEdecfAX4G+G/GXwv8O/tSXNzqdr46tYbjVY9LtbhBpZeymktY4yjo8+x0gXefM7mt/wAdfEmb4patbXesWdnYSRW0drDJa7/k+d3+ff8A79deZV6eBw0MLT9yf2v8RwZfRnmGKnip+/A4aOPy440T/Vp+7p1S3Fu9nPsm/wBZUVfCep95rbQ+Y/Dfiz4ieEP2vfFOpfC/R5tb8Vx3+qRwWkOn/bnaN/MRz5eOfkJr1uTxN+0h4j8fL4g+I/gPVNE0J5EfWtUvPC5soLa1RNkjvJ5fyJsT7/tS/sd/8pHNZ/3te/8ARMlfa3xW8ZXHjjx5q3wav7e3t/DXiGCHSrrUIXaO+SGaDe+zf8m//gFfreKq4ZUKVGvC/PGJ+T4eOJWNrToT+A+Sbf4g+FbgpHD4q0SSVhsRY7+He710P+rrtdV/4Jh/CnwVomoeI9O8SeNLi/0m1n1C1gnvrJ43eGN5kRwkG/76V8ZH9qrxtdNv/wCEL0uV5jvzHaXvz/8AkSvma/D3tf8AcHzep9PQ4gjti/cPUPj58QNV+GfgzT9U0aSGO6lv/sr+fAk6FNjv/GK+kPAfwj8K/Dj4PeFPj7oFg8HxN1fSLK/utTnuZJ4HnvcJdf6N9z/lpJ/ufhXzn+zPo2n/ALeHiTXPBvjaCbw7pmiWJ1iKTw44jnebz44dj+f5nybJ3r6N8ceNLrwx4Sl+DtnZ20nh7w00Gj2mo3Dv9qeO18vY7/wb/wDgFdjislwPsp+5WOR1P7WxvNT9+lA4vxR4kv8Axprt1rOrPC+o3OzfJBDsj+RNifJWTRVmzs3vJNifu40/ePJJ/BX563OpO7Pu0lSp2SPnL9r7TH1SLwVGjqqqdQeaVv8AVxJmD569U8dfEz9sHx9pVjpt58MNak02zkWeB08FmORykeze/B/gevKf22roHTfBNtb7o7MSXx5ODI/7j5396/VH4xfEa/8AhZ4H8P6hptnZ3097JDaOl9v8tE+ymT+D/rnX6fgq1HD5NQ9tDnh73/pR+ZY6lWrZtP2MuWfu/wDpJ8Haf420/S9NsrPxTrel6H4nhhVNS03ULpLWe2uc/vEkhx8n+5W7Y6lZ6rZpeafeW19bNwk1q+9H/wCB16rff8E8vhn8fL2X4ja/r/ijTtc8Vj+2760029slt4Z5/wB46J5kG/YM/wAdfLnx4169/ZR+J+o/DDwZoo13w1o8NrcWt9rCTT3befBHO4d4TGn35H/gryq+RUcVT9pgZ6v7j2MPnlSnPkxsD1G8ke3sb+dP9ZDazyJ/10SOR6yP2IPh/wCHP23YfGafFawOqr4Yn09NN/s+f+z9v2r7R55fy8b/APj1j/WvDNE/aY8Sa74l0vw9qHhrStPg1aZLF5EjuUdI53MLum+X/bevu7RvhXov/BP+7vbb4fyXviBPFcv+nHxNcI5g+xf6vy/IjT7/ANqffv8A9iujBYP+w6VStjIrn+wznzHG/wBqzp0MFL/Ec342+LnibxlF/ZOozWf2HTL3fa/Z7bY6bDJCnz/7lcrJGmqRyTwp5c6fft4/4/8AppHVS4k+0Tzz/wDPZ5JP++5PMqOOR7eSN0fy5E/5aV8FUxE6tS8z7ilh6dGn7Omcx8UP+SWeMv8AsFTfzryn9lT4qftFeDPhfq2k/Cfwhea/4Vm1Wd7q4t/Dn9o7bp4IEkj8zB2ny0h+T/br2f4tRpqnwn8ZzwosU6aTNJPAvR+f9Yleqf8ABKGb7J+yj4nnxv8AK8WahJ/3xp9i39K/QeHqnsMDWnb7Z8HxDDnxtGJ8++B9Y8c+GNPuY/jHpz+AnkKR6QuvWH9lfao03+f5e/Zv2Zg/7+V2Ol+KtF1+d4dK1vTtRnRN7x2Vyk0iJ/f+SvadV+Fehf8ABQDTbA/EKW88OjwnzZHwvPGgm+2pHv8AM8+N/ufZU+5/frx/9oX9mXwv+w/4Bs/Gvw/utZ8S6tqupx+H5rLxA8c8Kwukk+9PsyRuH3WqDr/E1ctTLMHmEXXw7tUn9nodVLNMVgZ+wxS+D7RZrzjTfEl34w/at0H4Q6t5U/g3Xbqysb22jh2TNDNBHI6ed/rE+f3ryqb9q7xhawvLN4Q0qKJfvu8F4g/Pza+s/wBn74I+H/GHwz0H9qa6uNStvHNjHdajHpdnKiaX5lk72sERR43m8t0iG8+Z3NXl+UTy2br42K5Lf+TE5jm0MfBUcHP3z0vXrCH9k+NPAfwxtotI8N6jC+sXUGpE30j3U37h/wB4/bZAleNxx/Z440T/AFaJ5ddb8RPiJe/EzXLXVL+ztrC4t7aO1RLXfs2b3f8Aj/365Svjsfip4uvz859PlmFjhKHK4ahXzD4Y8VfEXwb+2F4q1P4W6NPrPi5b7VEgtIdP+3P5b+Z5h8v/AHM19PV57+xv/wApHNb/AN/Xv/RElfVcLv2dWv8A4TweJ/4MP8Q+TxN+0j4n8dQ+IfiV4C1TQ9A3p/a+rXHhg2UFtaon7yR5PL/d/J/HXW6d448IXUazy+LNDMCttWOPUIfMmf8AuJX1n8VPElx448Y6t8JdUt7ez8L+IbaHSrnUbd2jvtk0G99m/wCT/wAcryLXP+CYfwr02zl1qPxR40kudNtXuraD7VY7E2JI6JsSD++ld1fCYHM5ym/ca7HmYfHYzL6cIz95focJeXD3E/zp5ez92kf9yvL/AI+fEDVfhv4JttX0aSGO6m1BLV/PgSdNmx3H3wf7leYN+1Z44u5i48GaVK8xL/u7W8/j/wC2lenfsyabYft3+LNX8C+N4pNC0zStOOtxT+G28udpkmjgCMZ/MTZsnk9K4cHw7iMLiIYjFRTpo9PFZ7hqlCdPDT98+lPAfwh8KfDf4N+Ffj9oFg0HxN1bSLG/utTnuXnhee92R3X+jfcx+8k/3PwrifFXirU/GmvT6zq7wyX0yIjyQQ7I/kTYnyV3PjrxldeF/Ck3wds7OGTw94ZEGjWuozu/2p47Xy9kj8bN/wDwCvMK8XOcb9Yr+zhP3IHpZNg/Y0fa1PjmFfPv7XWpJa23gyG7h8+zmkvfOTjd/wAsPnT0evoKvmv9tL/j08E/9dL7/wBoVfDf/Izo/wDb3/pIs/8A+RZW/r7R+lv7X7GTwz4V2K3/ACE5vT/nga+X/wCz4JJPMeBN/wDuD/GiiuXMf3lXmejNspfJR5d1qe7/AA1/aaPw78E6R4ei8N3d7Hp8Ow3EGpCFH+d3+5s/2q+W/j/8Rm+KX/BQb4FaydPuNKWK78PWvlTXPnt/yEnk3Z/7adPaiivpchxdaWLdKUrx5djwc5y/Dxwn1qMbTb3Pr79sa5aHXPCtvEflktr7e6gA/wCujr532t/db9KKK+XzaNsRzRdj3Mrk44NRQ3zE9f1b/Gvnv4/fHi102DxX4DOhSPPNCkA1JbsIvzmOb/U7PX/boor2+G8HRxGIjKqrtR5l63Wp5ud4ytTw8owlZN8vyPr7/gnSd37C3iwEMT9v1r7px/y5QV59FBHcwwboEl+T+NB/jRRRxNZ4jmjo/Inh1eyptL+tz0j4H/FWL4O6hrN9DpEupDUYIINlpdC22bHkf+5/00rjv28/2jD8Sv2d77Q30K806ZtXsn8641IXC/J5nbZRRXnZFmeJWLpUOb3b7WR7mcZRhKmFni3F+0fW7PUf2fBj/gmnoPcjQ7v7vH/MUnryDa391v0ooru4iipYjm2Z5fDXu0Kkd0W7a+guo/sVwwI3bIJgrZRvTr9z2rgfjF8S4/grplreX2ktqcst19kaCO5EBiOzfvL7H30UV5+TYWlisVTo11zROzNMVVwuHnVouzMv/gmn4iTxR+2d4q11IXtYNT0bVrtId25kDzxvsz+NfQn7QflzfGLxQkqb0zbfwD/n1g96KK+s4opq8KUW0l2PlOH6jjUlWteTOJ8PtDoXiTStSFmsp0++gujFHhJH2SRv9/8A4BX0pZftolryBR4Uvgu/H/IZH/xmiivzvC5licLJKlK3yPu8bluGxSvVV/mz46/4I/7m+JXxLyp3tokH8X/T2lemfEhi3xI8YbVZ/wDidXvp/wA93oor73ih80rrTbY+M4dgqNSaj/Wxg2Nv/aEqxKwVR952LEr+tR61rSWOhancW0cjWWm2090VGEeXYjvz/wB8UUV8BSk5VIxa3a6H3cpWi3Y+Ivjh8arf4vPopg0d9JOnpMj77rz9+98/3Er9hf2zMSeG9D27sf20/Yf88JKKK/Vc8oQw2XqjR91I/LcoxNTE5n7Ws7s+Tv7Pgkk3vAn/AHwP8a91+GP7Sw+G/gfTPDsfhm8vU09ZFNxb6mIEffO7/c2f7dFFfk1DG18NWvSlb5I/UsVgqGLpfvlf5s+Uv2kfiI/xS/b6+CmrNp9xpvk3Gg2n2e4uvPf/AJCTyZ3/APbSvsb9sr/kZvCmwZ/c333QB/y3joor9LzKo62U0pz+K25+eYOlHC5ty0tF2Pnra391v0o2t/db9KKK/LOd/wBI/SObyPn79oL46W2jR+Kvh82hyPdTW0MH9oC7CgbjHP8A6vZ7/wB+vsL/AIJs/J+xL4hPzE/2jrQ+U4/5dYP8aKK/Z6ODpYXJ4OktZxUn5u25+QYzGVcRmlpvS/LbyPMbaCO6tYN0CS/JH99BXovwV+KVv8HtY1fUotKk1L7fbJbBLSdbbZscv/c96KK/HcPjK1Gq5Qdvkj9lxODo16PLNXXqzlP29v2jD8TP2db7Q5dDvNOkfV7J/NuNRE6/J5h6bK9F/ZrjP/DsbQ1HLf2VqRGOP+YxcUUV+pYSvPF5NzVtWfluMwtPCZpTjR0Wmh5btb+636UbW/ut+lFFflPM7/8AAP0vm02OK+K/xQh+Eug2Wp3Oly6ql5cfZ/LF0INg2B/7j+tYn/BN7Vz4t/bT1rxMsJtbS8sdYvPJd97IH+fYf733+9FFfruSYajh8vlWhH3pbn5fnuKqVsXGlLZH0p+0UTdfGPxMjqzxRm1RE4/59YK4Lw/5WheI9L1KOzWQ6fewXXlx4SR9kkb/AH/+AUUV+dYyUoVpzg7Nn3mCfPhYQnqkfS1j+2ZLNfwKvhO/MbzIMPrAP8f+5XyL/wAEmbhpPit8VZpAVeTSI35bd/y+g0UV9nkeOr4rD4hV5c1tr9D5HNcvw9CdB042vv5nffEZmb4jeL8Kzn+2b30/57vXPbW/ut+lFFfnmIqN1G9PuR95hZWw6VipqF8dN0fUtQ8ozR2NpPeGNcJvCI77P/HK+L/jn8YovjB/Yhh0Z9JGmJMrBrsTmTeynOdiY6UUV+hcI4elLnxEo3nF2TfTQ+M4grVJJYe9on6/ftiMsnhHwzt3PjVO4H/PrJXy9aNbw70ntUnt5/vxlB/jRRXzOYe/W5tmfQ5RFQwfJuj3T4f/ALRqfC3wTo3h218P3d/a26OYrmHUBAku+d3+5s+T79fJn7SHxMb4sft3fA3V0sJdNeGXRbIwy3X2hyRqsz7t4VP+elFFfQ5Diq0sW6UpXjy7Hz2a5fh44T61GNp33PsD9sxt3iPwpsGf3N990Af8to6+fNrf3W/Siivl82jbEc0XY9/K5uGDUUG1v7rfpXz9+0F8dLfR4/Ffw+bQpHupraGAagLsKPmMc/8Aqwn/ALPRRXt8N4OjiMRGVVXajzL1utTzs8xlanh5RhKyb5fkfW3/AATKbH7FfjQEMT/bmrfdOP8AmHWlee20Md1awboEl+SP76D/ABoorXiazrXWnoYcOxVOm0j0/wCB/wARIPg3dazrkemNfx30MdgkdnMLZg6Pv3Z2cL7VzH7cv7SzfEz9m3xDox8P3dg019YMLiXURKnyTb8bNnt/KiivJyPM8SsZSoc3u32sj3c2yjCVsLPFyj+8fW7O3/ZpjH/DsbQ06v8A2VqX3eP+YxcV5btb+636UUV6HEMVUxLbPM4a92g476htb+636VxPxY+KUPwk0Gy1O40uXVkvLj7N5YuhBsGzf/ceiivHyXC08VjoU63vI9PNsVVwuH9rRdmYH/BO7xRH4u/bf1PXUtmtYNSsNYu1hL7mQOhfZn8a+k/2hGST4zeKEYb0za/wD/n1goor7XiinpClFtJdj47h+o41pV7XkziPD7Q6D4k0rUks1lOnX0F0Yo8JI+ySN/v/APAK+l7P9sx3voE/4RHUNkkyD/kMjH3/AE2UUV+d4XMsThXalK3yR+hY3KsJilerH8WfIn/BJidpfip8VXcFfM0eOT5m3f8AL6DXoXxOYv8AE7xpsU/8hm69P79FFfbcTPnae22x8Jw/TVGc1H+tjmtrf3W/Sqt9enTtJ1LUPKM0dlaTXhRcJvCI77P/AByiivgKUuapGLSs2uh93KVot2Pi/wCOHxstfi9HoqWujS6QNO87f5l4Ljfv2c/6tMfcr9fv2xMN4R8MY3P/AMTMdQP+fWSiiv1nPKEMNl6o0fdSPyvKcXVxOY+1qu7PlL7BA8nmSQI7/wC4K92+GP7S3/CufA2l+HY/Dl5fDT0dDcW+piBH3u7/AHNn+3RRX5NQx1fDVv3crfcfquKwVHF0rVlf5s+V/wBpL4k/8LW/bw+Bmr/2fPpjQy6JZGGe7E78arM+7ft/6aV9l/tlzG31TwvDCcpINR37QB/y0goor9OzKXtsppznq7bn53gKUaGbcsNl0Pm/a391v0o2t/db9KKK/Kud/wBI/SObyPAvj38coNEbxT4BOgyPdzWyQ/bxehQN6xzZMew55/26+s/+CZK7/wBivxovzMRrurn5Tj/mHWlFFfstHCUsLk6dJWc4qT83bc/IsZi6uIzRxm9IbLsee21ul1awbo0l+SP76D/GvRPgr8Urf4Paxq+pRaVJqX2+2S2CWk622zY5f+570UV+PYfGVqNbmg7fJH7HicHRr0LVFf5swP26P2kn+JP7NHiHRX0C705p72y/0i41ITp8k+/7m32r0D9mWNv+HZmhqOW/sbVSMcf8xS4oor9Tw1eeIyjmq6v/AIB+W4zC08JmkI0dFpp8zyba391v0q3bX0F1H9iuGBG7ZBMFbKN6dfue1FFflfO77L7j9I5tNjg/jB8Sovg1pltd3+lNqUktz9kaCO5EBjOzfuL7H31zX/BOzxRH4q/bc1XxAls9rBqVjrF2kJfcyb037M/jRRX69lGCo4XAyq0170tz8yzvFVMTi40p7I+lf2ho43+NHimNxvjza/wD/n1g964fw/5WheI9L1KOzWQ6fewXXlxgJI+ySN/v/wDAKKK/M8ZKUK05Qdmz7zBy9phYQnqkfStl+2c81/Ai+FNRMbzIMPrAP8fptr5D/wCCTNw0nxT+KTyAr5mjxv8AM27/AJfQaKK+2yPG18Vh8Qq8ua21+h8jmuXYehOg6cbX38z0X4nsW+J3jPCs/wDxObr0/v1z1jb/ANoSrErBVH3nYsSv60UV+d1KjddvT7kfa4Z8uHSSDVdSWz0y9a2iLWtnC90wXCGXZG78/wDfFfEHxw+NNt8YP7DjttGl0n+zjNvEl2LjzN+zn7iY+5RRX6Hwrh6U5VMRNXnDRN9L3PlOIK1Rxp0E7RnufsF+2BtbwP4f27nP9rx9QP8An1kr5P8AsEDyeZJAjv8A7goor5nMn7StzvRntZS3Tw3Ktj3T4XftJN8NfBGmeHovD15erZ+eTPBqQhRt87v9zZ/t18k/tW/Exvi1+2j8HdWXT5NOaGLSLIwy3X2hyRqU0m7ftT/npRRX0WQYutLFOlKV42eh5Gc5fh44P61GNpt7n2v+2kS+u+FNh7aj90AfxwV857W/ut+lFFfLZvG2IcouzPYyuThg6cUG1v7rfpXgnx8+OEOh/wDCU+AW0GWa6ms44P7Q+3BQN6RzZ8vZ6/7dFFe1w9hKOIxEZVVdqPMvW61MM/xFShh5RpOyZ9b/APBLvD/seeLPvE/8JBq3Q4/5h1pXm1jBHdWNp5sCS/uI/voP8aKK6OJnetdaehxcLxXI1LX19Weh/BP4nRfCHxJqt/DpTajLe2UdqIbScWxT95v+/srD/bl/aQb4lfs0+INFbw/d6e0+oWTC4uNSE6fJNvxs2UUV4uR5liVjKVDm9260se5nOUYSWHnW5fefW7O4/ZdU/wDDs3RkC5b+ytVxg4/5itxXlu1v7rfpRRXpcQQVTFNs83hv3KDjuG1v7rfpXGfFb4lRfCnw/bardaU+q/aLz7MYRciFkG12+/sfuoooryclwtLFYxUq65onp5hiKlHDOrB2kYn/AATl1weLv21tY8TrbNa2t7YaxdmF5PMZA679p/vff7+lfSX7R9wt58YvEKynfGkVt+7y3/PqnvRRX2XFENIUYtqK7HxnD83GtOva8mee6K1vpOtaZqcVqhWzu4LrdGgR/kk3+v8AsV9NWX7Zu++gT/hEtS2PMg/5DIx9/wBNlFFfnOFzHE4WSVKVvkfeY3LcNileqr/Nnx7/AMEs7trn9o74q3MgK+dok0nzNu66jBXpnxQYt8T/ABptVn/4nV16f36KK+44mfO1LZ6bHxmQU1RqzUf62OZ2t/db9Kgv7o6fp93dPEZI7WGecpx8+xHf/wBkoor4CnLmmk0t+x91KVk3Y+Lfjh8a7X4vx6Klro0ukf2aJg/mXguN+/Z/0zTH3P1r9eP2rH834Z+ENu5h/aEHUDr9hkoor9azyhDC5f7Gj7qR+VZTiquJzP2tV3Z8xWjW8O9J7VJ7ef78ZQf417l8O/2jIvhT4J0nw9b6BeahbQ+f5V1BqAgWXfO78Js+T79FFflWGx1ej8EvwR+p4zA0MRSSqK/zZ8hftYfE7/hbX7Z/wc1kWD6cYo9IszDLcfaHyNUnffvCp/z0r7V/bSX/AInvhQxnr/aP3QB/HBRRX6LmM3iMnpSqau25+fYKlHC5ry0tEuh857W/ut+lG1v7rfpRRX5dzv8ApH6RzeR4J8f/AI4waI/ifwAdBkmu7iygh+3/AG0KI96Rzf6vZz1/v19Yf8EtGA/ZB8bDJJ/4SXUzwB/0DbSiiv2ihg6WFyiMqSs5xUn5u25+R4rFVcVmko1HpHZdjz6xgjurG082BJf3Ef30Fem/BX4gW/wb1DVdfGltex30MdgkdrOIWD79+7Oz7vtRRX45h8ZWo1rwdvkj9bxGGp16CVTX5swP23P2mJfiP+zD4w0V9Bu7D7VcWA+0SaiJU+S6R/ubf9iux/ZfU/8ADsvR8Lkf2TquMHH/ADFLiiiv07D4mriMm56ru7/ofmmMw1PCZrCNHRaafM8s2t/db9KNrf3W/Siivy3md/8AgH6XzabHF/Fb4lRfCTw9b6tdaW+qfaLz7MYBciFkG12+/sfuorkP+Ce/idPFf7cx1yO3azg1KHWrtId+9kDwSPsz7UUV+vZHhaNDASrQj70r3Py/PcVUrYuNKWyPqX9o3y5vjP4hjxv+Wy6oP+fVPevP9FNvpOuaZfLaozWd1BdbYkCSfJJv+/n/AGKKK/OcYuStOcHZs+8wMufCwjPVI+m7X9syWS+gX/hE7/y5JkGDrA/v/wC5Xx3/AMEtbxrj9oz4pTSBlebRJ5Pmbd/zEYDRRX2eR46viqWIVeXNba/Q+RzXLsPRnQdONr7+Z6Z8TmZvih4zkCl/+Jzc+n9+ua2t/db9KKK/O69Ruu3p9yPucK7YdJLoQahdNp+m3l06tJHawTzmPj59kcj/APslfFvxw+NVv8X49DhtdFm0j+zTNu8y9Fx5m/Zz/q0x9yiiv0HhHDUpyrYiavODsm+lz5DiCtU5aeHTtGe5+u37VjK3wx8IFdzn+0IOoH/PjJXyz9ggeTzJIEd/9wUUV83mn7ytzPRnrZPJ08Nyp6Hunwu/aVb4a+B9M8PReHry9Fn55M8GpCFG3zu/3Nn+3Xyh+1Z8Qm+Lf7a3wZ1RNMl0x0i0iy8ma6+0Pn+1J337xs/56UUV9FkGLrSxTpSleNtjyM5y/Dxwn1qMbTvufbX7Zl40V54ahiP7uSXUN+0Af8tIK+adrf3W/Siivl83XLiXKLsz28pk4YJRQbW/ut+leD/Hj47Q+GZ/E/gVtAlmuLizSFb/AO2hQnnQJJ/q9nP3/wC/RRXtcO4OjiMTzVVd25vndamef4mrh8Oo0nZM+q/+CWzhf2PvGgbOf+El1M/KB/0C7SuAsbdLrTbTzY1l/cR/fQUUV08Tu9a609Dz+HY+zpu39bnofwT+J8Xwg8Sarfw6U2oyX1ilqIbScWxT95v+/srL/be/aYPxE/Zj8WaD/YF7p32q4sFFxPqYnTCXKPjZs/6Z0UV4eRZliVjKVDm9260sj3s5yjCVMLPFuL9o+t2dv+ynHu/4Jn6Ui/M/9i6z1/7CN3Xku1v7rfpRRXpcQwVTFNs8rhtctBx3Rbtr6C6j+xXDAjdsgmCtlG9Ov3PauC+MHxCT4O6LFfXelvqTG++xvAtyIWU7HfeX2Pv+5RRXmZPhaWKxUKVdc0TtzTFVcLQnVouzOR/YG8XR+Nf26l19LVrKHUINYukhL72QPayfLmvqT9opkuPjN4hRhuTbZ/wD/n1SiivreKKaVOFGLaiux8rw/NxrSr2vJ/8AAPPdFa30rVtM1NbVGWzuoLrdEgR/kkjf1/2K+mrL9szffQJ/wiWpbHmQf8hkY+/6bKKK/OsLmWJwrSpSt8j7vG5bhsUr1Vf5s+Pf+CWd01x+0Z8U55AyvPok0nzNu/5iMFen/Fxifi140ZVLJ/a03XFFFff8RP2sPe8vyPi+H6ao1Z8v9bHNWNv/AGhKsSsFUfedixK/rRrGpLaabeG2iLW1hDPdELhDLsR35/74oor89pzcppNfgfdylaLdj4h+OHxutfi9DokVto02kDTfO3eZeC48zfs/6Zpj7lfrv+1Y+74Z+ENu5x/aEHUD/nxkoor9Xz3D08Pl6o0fdSPyrKcXVxOZ+1qu7PlmSwgkk3vAn/fA/wAa9y+FP7Sy/DHwPp3h6Pw9d332J53Nxb6gsCPvnd/ubP8Aboor8lw2Nr0avuS/I/VsVgqOLpWqq/zZ8gftpfEr/ha37WHww1ZtNl0ow2OnWfkz3QuHyNRnffv2/wDTT9K+5P2zFH9qeFyBn97qg+UAf8tIKKK/TMyqOtk9OU9Xbc/OsHSjhc25aWi7Hzjtb+636UbW/ut+lFFflnO/6R+k83keEfHr45W/heXxJ4D/ALCkmurmwSH7f9tCgeakcmfL8vn/AL7r6y/4JVKG/ZJ8Xgkk/wDCUameAP8AoHWNFFfs+HwlLC5PGVJWc4qT833PyTG4qris0kqj0i9F2POLKCO50203QJL+4j++g/xrvfgr8SIfhH4q1DUotJbUpLqz+wCC0nW2Knej79+z/Yoor8dpYytRrNwdvkj9gxODo16HLUV/mzO/bg/aXb4ifsx+K9COg3mnC6uNPUTzakJ0+S63/c2/9M/5V2/7KMY/4dl6co5f+y9c6cf8vtxRRX6nhK86+T89XV3Py/GYWnhM1hGjotNDyja391v0o2t/db9KKK/KOZ3/AOAfpXNpscf8WPiRD8J/DcGrXelvqm+8SxMAuRC33Hffv2P/AHK5n/gnzrh8cftxXHilLb7FaXUGr3nks/mMgkgk+Qn+LrRRX67kmGo4fAOvCPvS3PzHPcXVr4uNGXwo+of2mj9q+MetI6s8SWVkmzj/AJ415lpq29jq1hex2qH7LdQT/u0Cfck3+tFFfnmMvCtOUHZs+6y2XNhYRlqkfT8H7Zksl9An/CJ3/lvKgx/bA/v/AO5Xxb/wS3vWuf2n/iVNIGXzvDV3J8zbv+YjaUUV9jkWOr4qliFXlzW2v0Pkc1y7D0Z0HTja+/meu/Fpmb4seM3Clk/tabrj0rl9rf3W/SiivzuvUbrt2X3I+5wrth0kuhHcMYLWeZ1b9zA8/b+CPfXxV8b/AI12/wAX00KK10SbSf7NM27zL0XHmb9n/TNMfcoor9B4Rw9Kc62Imrzg9G+lz5HiCtUcaeHTtGe5/9k=)

### Syobon Action (Cat Mario)

It is a parody of Super Mario Bros, created in 2007. The player controlling a cat must travel through a series of traps. The player has to memorise the traps and dodge them accordingly.

Diagram

Description automatically generated

### Game Analysis

Both of the games have minimal graphics with little to no animations. It shows that the quality of animation or graphics is not necessary as long as the game mechanics are fun to play. Both games move the camera when the character leaves a specific boundary on the screen. In both games, the player can only increase or decrease the character’s speed instead of directly changing the character’s speed. Although it is more realistic to do it this way, the player will have a more challenging time picking it up. Both games have a life system in which the player can see how many lives they have left. Both games have a single jump system where double jumps are not allowed.

While Super Mario Bros has shorter levels, Syobon Action has longer levels with a checkpoint system where the player can respawn in the checkpoint after they die.

While Super Mario Bros is proven to be enjoyable to play, it has been out for so long that everyone has already gotten used to the play style. On the other hand, while Syobon Action has much more mechanics involved and much more features, the game is too complicated that it becomes frustrating to play.

### Current System

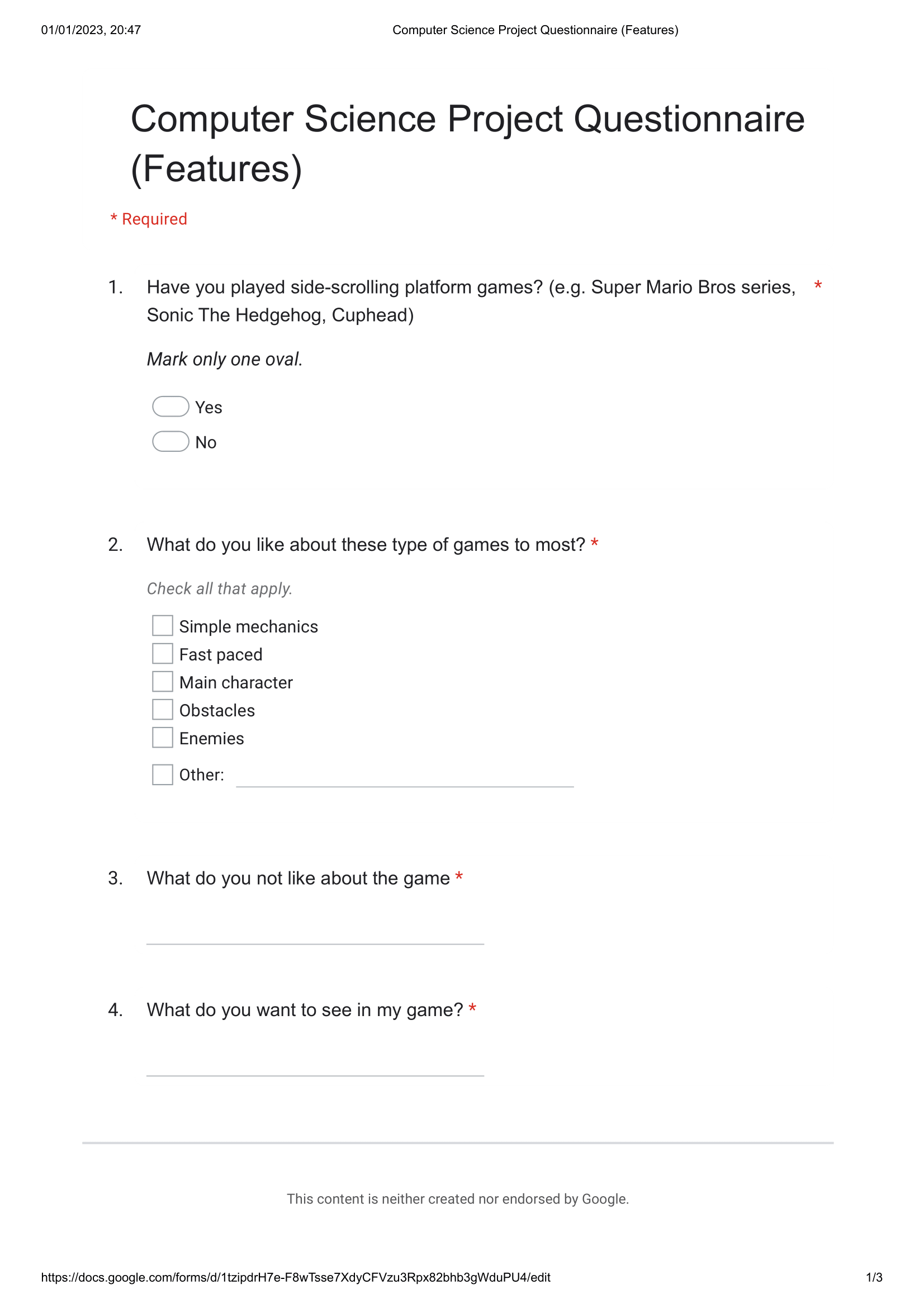
Diagram

Description automatically generated

## Investigation

### Questionnaire 1 (Features of the Game)

In this questionnaire, I would like to know what features the end user like or not like.



### Result of Questionnaire 1

Have you played side-scrolling platform games? (e.g. Super Mario Bros series, Sonic The Hedgehog, Cuphead)

* Yes (9)

What do you like about these type of games to most?

* Simple mechanics (8)
* Fast paced (6)
* Main characters (2)
* Obstacles (7)
* Enemies (2)
* Music (1)
* Graphics (1)

What do you not like about the game?

* Repetition (2)
* Too difficult (4)
* Low resolution (1)
* Levels too short (1)
* Too easy (1)
* No checkpoint (1)

What do you want to see in my game?

* More obstacles (2)
* Easier controlled movement (2)
* Better level design (1)
* Longer levels (1)
* More checkpoints (1)

Through the questionnaire, I gather the opinion of people in age group 16-25 years old on the features of side-scrolling platform games.

We can see that they mainly like side-scrolling platform game because the simple mechanism, fast paced and obstacles elements. They do not like about the repetition and think some of these games might be too difficult. They want to see smoother movements and more obstacles.

Specification:

* More obstacles
* Less repetition
* Lower difficulty
* Smoother movements

### Questionnaire 2 (Detailed Specification)

To investigate the detailed specification the user wants

Table

Description automatically generatedText, letter

Description automatically generated

### Result of Questionnaire 2

Window size:

* Full screen (4)
* Windowed (5)

How large should a tile be?

* 20x20 pixel (2)
* 30x30 pixel (6)
* 40x40 pixel (1)

Size of the character:

* 1x2 tiles (3)
* 1x1 tiles (2)
* Smaller than a tile (4)

More traps or enemies:

* Traps (6)
* Enemies (3)

Do want a boss fight:

* Yes (4)
* No (5)

Do you want any checkpoint:

* Yes (8)
* No (1)

How many lives should the player have?

* 1 (1)
* 2-3 (3)
* 4-5 (1)
* Infinite (4)

From this questionnaire, I have made the following specification:

* Windowed screen
* 30x30 pixel a tile
* Character smaller than a tile
* More traps
* No boss fight
* Have checkpoints
* Infinite Lives

## My Approach

I will use Python Module Pygame as the main framework for my game. It is a Python Module set designed for making video games in Python. The game is uncomplicated, so the slowness of Python and Pygame will have limited effect. On the other hand, Python and Pygame are much easier to pick up and require less development experience. I will use it to design the game and user interface. I will mix up existing tile maps for the game. I aimed to design a game that is easy to pick up but fun and challenging to play.

To keep it simple, the game mechanics will only consist of moving horizontally, jumping and shooting horizontally. I will introduce some new enemies and traps to add some different play styles to the game. The game will contain longer levels to smoothen the gaming experience. To accommodate the longer levels, I will add checkpoints to the levels to reduce the frustration after dying.

The game will not have a life system where the player must start again after they die a certain amount of times. Instead, it will have a death system where the game will only record the number of deaths the players have but will not force a restart after a certain amount of death, allowing the players to try as many times as they need. They will not lose any progress even if they have died multiple times.

The game has a user interface where players can access the control menu and all the levels.

## Limitation

The user’s machine must be able to download a Python3 interpreter and install the Pygame library. The user’s machine needs about 5 MB of free space to download the code and assets of the game. The player with colourblind might have a more challenging time playing the game. Text and blocks are designed with high-contrast colours to better the gaming experience of people with colour deficiency. The size of the fonts is bigger to make the text stand out from the background. The user’s machine must connect to a keyboard, and other input devices like controllers or joysticks are incompatible. The screen of the user’s machine must be at least 810 x 540 px in size to fit the gaming output.

## Solution Requirements

### Hardware

|  |  |
| --- | --- |
| Requirement | Justification |
| 30Hz monitor | Allow the animation of the game to run smoothly |
| 5MB free space on hard disk space | To store the assets and code of the program |
| Standard Keyboard | An input device for the game |
| Mouse / Mouse pad | An input device for the game |
| 4GB RAM | System requirement to run Python |
| x86 64-bit CPU | System requirement to run Python |

### Software

|  |  |
| --- | --- |
| Requirement | Justification |
| Python3 interpreter | To run the Python code |
| Pygame library | The program contains code loaded from the Pygame library |
| Windows 7 or later version or  Linux: RHEL 6/7, 64-bit or  Mac OS X 10.11 or higher, 64-bit | System requirement to run Python |

## Success Criteria

1. Design requirement
   1. The screen is 810 x 540 px in size
   2. Each block is 30 x 30 px in size
   3. Character smaller than a tile
   4. Each map is at least 18 x 80 blocks in length
   5. There will be buttons for the user to click on to access the game levels
   6. There will be a control menu showing the controls of the game
   7. The game will show a death screen when the game ends
   8. The game show a victory screen when the player wins, showing the time used and death number
   9. The game will return to the main menu whenever a game ends
   10. The game will have a ranking screen showing the top entries of each level
   11. Control button
   12. Exit game button
   13. Ranking board button
   14. The game can be restarted (Restart button)
   15. Images showing the start of the level
   16. A SQLite table to store all the successful entries
   17. Flat file to store the information of enemies and traps
2. Input requirements
   1. The user can use a keyboard as input to control the character
   2. The user can use the mouse pointer to navigate between levels and the menu
3. Game requirement
   1. The character can move left and right
   2. The character can double jump
   3. The character can shoot horizontally
   4. Sky blue background
   5. Ground tiles
   6. The camera will follow the character whenever it leaves the screen
   7. The character can shoot the enemies, and enemies will reduce in health when the bullet hits them
   8. When the character falls off the map, the character dies
   9. When an enemy or trap hits the character, the character dies
   10. Whenever the character dies, the death count increase by one
   11. Infinite lives
   12. The character respawns at the checkpoint after it dies
   13. There will be different traps and enemies
   14. More traps
   15. There will be a flag pole that the character needs to touch to win the game
   16. The camera will stop when the character reaches the boundaries of the map
   17. Timer for the game
   18. Show death count
   19. Can access menu within the game

# Design of the solution

The game will be written using Python with Pygame library. The player will be able to control a character through a side-scrolling platform. The player’s objective is to reach the flag pole at the end of each level using as little time as possible and die as few times as possible. There will be enemies and traps throughout the level. When the player is defeated, it will display a game-over screen. It will respawn at the checkpoint, and a death count will be added. A victory screen will be displayed when the player reaches the end and wins. The player’s name, the time taken and the death count will be recorded. The top 3 fastest-finishing records of each level can be seen on the ranking board through the main screen.

## Problem decomposition

1. Main Screen
   1. Display
      1. Screen: 810 x 540 px
      2. Background colour: Brown
      3. White level buttons
      4. Black menu button
   2. Level button
      1. Black text
      2. White background
      3. Level number
   3. Main Menu
      1. Green background button with black text
      2. Instruction button
      3. Ranking board button
      4. Quit game button
   4. Ranking board
      1. Green background
      2. White boxes
      3. Black text
      4. Fetch data from a local SQL database
      5. Data is written to the SQL database whenever they finish the level
      6. Column
         1. Name
         2. Time taken
         3. Number of deaths
         4. Level
      7. Close button
   5. Instruction screen
      1. Green background
      2. Black text
      3. Clearly stating the button and its uses
2. During the gameplay
   1. Map
      1. Ground tiles
      2. Grass Tiles
      3. Checkpoint
         1. Change appearance when it is activated
         2. Save the coordinate as the respawn point when it is activated
         3. Not a solid block (Cannot interact with the player)
      4. Flag pole
         1. Can be touched to win the game
         2. Will activate winning procedure when it is touched
         3. The game will end after the winning procedure
      5. The player dies if he falls out of the map
   2. Character
      1. About 5 x 5 px in size
      2. Can move left or right at a constant speed
      3. Can jump and double jump
      4. Constant download acceleration will be applied to mimic the gravitational acceleration
      5. Can shoot bullets
         1. Bullet
            1. Will be destroyed if it hits an enemy, a tile or the edge of the map
            2. It will reduce the health of the enemy when it hits them
            3. Will not interact with traps
   3. Enemy
      1. Goomba
         1. Will patrol within an area
         2. It will kill the player if it is touched
         3. Can be killed by the bullet
         4. Show the health of the object
      2. Traps
         1. All the traps will be loaded into the game through a comma-separated file
         2. It will contain the type of trap, the base of the spike, the height of the spike, the coordinate, the activation zone
         3. Spike
            1. Made up of small rectangles in a triangular shape
            2. It will kill the player if it is touched
            3. It can have a customised base and height
         4. Moving spike
            1. Same with spike
            2. Will move in any direction
            3. It is activated when the player reaches certain zone
         5. Growing spike
            1. A spike that can change in size
            2. Will change in terms of the ratio of the base and height
            3. It is activated when the player reaches a certain zone
         6. Horizontal spike
            1. Made up of small rectangles in a triangular shape pointing horizontally
            2. It can have a customised base and height
         7. Horizontal moving spike
            1. Same with horizontal spike
            2. It can move horizontally, vertically, or diagonally
            3. It is activated when the player reaches a particular zone
         8. Invisible block
            1. It will appear invisible
            2. It will be activated when the player reaches a particular zone
            3. It will appear when activated
         9. Fake block
            1. It will appear like a typical ground tile
            2. It will disappear whenever the character touches it
   4. End screen
      1. Death
         1. Show “Defeated”
         2. Restart the game at respawn point
         3. Add one death count
      2. Win
         1. Show “Victory”
         2. Return to the main menu
         3. Insert a record into the SQL database with the name, time taken, the death count and level number
   5. Game board
      1. Show the time taken
      2. Show the death count
3. In-game menu
   1. Instruction button
      1. Change to the instruction screen
      2. Show the button and their function
   2. Restart button
      1. Restart the game in the initial spawning point
      2. Reset time count to zero
      3. Reset death count to zero
   3. Quit game button
      1. Interrupt the game
      2. Return to the main screen

## Initial User Interface Design

### Main Screen

A picture containing diagram

Description automatically generated

* Menu Button
  + Top right of the screen
  + Used to access instruction and ranking board
* Level Button
  + Contain an image showing the initial state of the level
  + A play button to initialise the game
* A progress bar
  + Show the progress of the player

#### Success Criteria

* Buttons for user to click on to access the game levels

### Menu

Graphical user interface, application

Description automatically generated

* Control (Instruction) Button to navigate the instructions
* Rank button to navigate to the ranking board
* Quit button to quit the whole game

#### Success Criteria

* Control button
* Ranking board button

### Ranking Board

Chart, bar chart

Description automatically generated

* Entries will be loaded from a local SQL database
* Entries shown in the middle of the screen
* Close button to return to the main menu

#### Success Criteria

* The game will have ranking screen showing the top entries for each level

### Instructions Screen

Graphical user interface, text, application

Description automatically generated

This is the instruction screen

* Instructions are shown in line in the middle of the screen

#### Success Criteria

* There will be a control menu showing the controls of the game

### In-game menu

Graphical user interface, application

Description automatically generated

* Controls button (navigate to the instructions screen)
* Restart button (restart the game)
* Quit button (Quit to main screen)

#### Success Criteria

* There will be a control menu showing the controls of the game

### Gameplay Screen

Timeline

Description automatically generated with low confidence

It is the screen when the player is playing the game

* Player (red circle)
* Enemies (grey circle)
* Ground tiles (Green and brown rectangles)
* Menu button
* Death count
* Timer

#### Success Criteria

* Timer for the game
* Show death count
* Can access menu within the game

### Win Screen

Timeline

Description automatically generated

* The win screen will be displayed when the player reaches the end and wins
* The time taken and the number of death will be shown

#### Success Criteria

* The game show a victory screen when the player wins, showing the time used and death number

## Final Design

### Main screen

Chart

Description automatically generated

The concept of the main screen is similar to the design. The progress bar has been removed. There are only two levels, and it will be excessive to add a progress bar. The menu button is replaced by a symbol.

#### Success Criteria

* Images showing the start of the level
* Buttons for user to click on to access the game levels

### Rank Board

Table

Description automatically generated

It is the ranking board.

The top 3 best wins will be shown for each level. The ranking is based on the shortest time and then the fewest death.

The name entered when the game is initialised will be used on the ranking board.

The text will be black and white, and the background will be blue-green.

#### Success Criteria

* The game will have a ranking screen showing the top entries of each level

### Gameplay screen

A picture containing graphical user interface

Description automatically generated

It is the gameplay display

* The background is sky blue
* The ground tile is brown
* The timer is shown on the top left corner
* The death count is shown on the top right
* The menu button is shown on the top right corner of the screen

#### Success Criteria

* Sky blue background
* Groud tiles
* Death count
* Timer
* In game menu button

Chart

Description automatically generated

This is the game pause screen. It overlays the game. Everything in the game will stop running. When the paused screen is closed, the game will continue.

The user can restart the game with a new timer and death count. They can also access the instruction menu or exit back to the main screen.

#### Success Criteria

* Restart button
* Exit game button
* Control button

## Class Diagram

Diagram

Description automatically generated

Diagram, engineering drawing

Description automatically generated

Diagram, engineering drawing

Description automatically generatedChart

Description automatically generatedDiagram

Description automatically generatedDiagram, engineering drawing

Description automatically generated

Logic FlowDiagram

Description automatically generated

## Key Data Structure Design

### Main

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Data Type | Variable | Description |
| Current Screen | String | currentScreen | The screen it should be displaying currently |
| Instruction Screen | Customised class | instructionScreen | The instruction screen and its attributes |
| Menu | Cutomised class | Menu | The setting menu in the main screen |
| Player Name | String | Player\_name | The player’s name |
| Status | List | Status | The status / action it should be doing |
| Ranking Screen | Customised class | rankingScreen | The ranking board screen |
| Font | Pygame Font | Font | The font type that will be used to render text |

### Game

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Data Type | Variable name | Description |
| Camera | Customised class | Camera | The camera of the game |
| Clock | Pygame time clock | Clock | The clock that the game cycle on |
| Player | Player | Player | The player character |
| Current Screen | String | currentScreen | The screen that should be displayed currently |
| Checkpoint | Tuple | Checkpoint | The coordinate that the character should be respawn on |
| Status | List | Status | The status / action it should do |
| Trap Group | Pygame sprite group | Trap\_group | The group of traps |
| Restart | Bool | Restart | Restart the game or not |
| Menu Screen | Customised class | menuScreen | The menu screen in game |
| Screen | Pygame surface | Screen | The screen the game should display on |

### Player

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Data Type | Variable name | Description |
| Horizontal Speed | Int | xSpeed | The horizontal displacement per frame of the character |
| Vertical Speed | Int | ySpeed | The vertical displacement per frame of the character |
| Bullet group | Pygame sprite group | Bullet\_group | Group of bullet the character has shot out |
| Bullet Timer | Int | Bullet\_timer | The cooldown of the shooting |
| Max horizontal speed | Int | Horizontal\_max\_speed | The maximum horizontal speed the player can travel in |
| Face Direction | Int | Face\_direction | The direction the character is facing |
| Collision box | Int | Rect | The collision box of the character |
| Health | Int | Health | The health point of the player |
| Image | Pygame display surface | Image | The display image of the character |

### Map

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Data Type | Variable Name | Description |
| Checkpoint group | List | Checkpoint\_group | A list of checkpoint tiles |
| Dead Zone | List | Dead\_zone | A list of dead zones |
| Finish Point | Customised class | Finish\_point | The flag pole where the player will win the game if it reaches it |
| Ground Sprite Group | Pygame sprite group | groundSpriteGroup | The group of ground tiles |
| Tile Group | Pygame sprite group | tileGroup | The group of all tiles |

## Algorithms

### Game

#### Read status

for stat in status:

    if stat == status1:

        # Action 1

        pass

    elif stat == status2:

        # Action 2

        pass

    elif stat == status3:

        # Action 3

        pass

    elif stat == status4:

        # Action 4

        pass

    elif stat == status5:

        # Action 5

        pass

    # endif

# next stat

It will identify the status code in the status and do related actions

#### parseEntities

It parse all entitles of the game from a csv file

def parseEntities(self, filename: str):

    with open(filename, 'r') as f:

        reader =  csv.reader(f)

        next(reader)

        for row in reader:

            self.create\_enemies(row)

    self.gameSpriteGroup.extend(self.enemy\_sprite\_group)

#### keyResponse

def keyResponse(self: Game, event: pygame.event):

        # Different response with respect to the screen mode

        if self.currentScreen == screen1:

            self.status = self.screen1.keyResponse(event,self.status)

        elif self.currentScreen == screen2:

            self.status = self.screen2.keyResponse(event, self.status)

        elif self.currentScreen == GAME:

            self.status = self.player.keyResponse(event, self.status)

        self.gameResponse(event)

It passes the Pygame event into the procedure and further pass it to the screen procedure according to the screen it is displaying.

### Player

#### Character control

def keyResponse(self: Player,event: pygame.event, status: list) -> list:

        if event.type == pygame.KEYDOWN:

            # If W is pressed, jump

            if event.key == pygame.K\_w:

                self.jump()

            # If space is pressed, shoot bullet

            elif event.key == pygame.K\_SPACE:

                self.shoot\_bullet()

        # get the key being pressed

        key\_p = pygame.key.get\_pressed()

        # If the A is pressed down, set speed as left

        if key\_p[pygame.K\_a]:

            self.set\_speed\_x(-self.horizonal\_max\_speed)

            self.face\_direction = -1

        # If the D is pressed down, set speed as right

        elif key\_p[pygame.K\_d]:

            self.set\_speed\_x(self.horizonal\_max\_speed)

            self.face\_direction = 1

        # If None of A or D is pressed, set speed to 0

        else:

            self.set\_speed\_x(0)

        return status

It check the input key of the keyboard. If the user pressed “W”, it will jump. If the user pressed Space bar, it will shoot. If the user pressed “A” or “D”, it will move to either direction

#### Shoot Bullet

def shoot\_bullet(self: Player):

    # Check if the player is allow to shoot bullets

    if self.bullet\_available:

        bullet = Bullet(self.face\_direction, self.rect.center)

        self.bullet\_group.add(bullet)

        self.bullet\_available = False

It check if the player bullet is available. If yes, it will create a bullet object facing the direction the player is facing.

#### Interaction with enemies

def enemy\_interaction(self: Player,enemy\_sprite\_group: list, trap\_group: TrapGroup, trapClasses: tuple) -> str:

        # Player dies if it touches with any enemy

        for enemy in enemy\_sprite\_group:

            if type(enemy) == Goomba:

                if pygame.sprite.collide\_rect(self, enemy):

                    return PLAYERDEATH

        # Player dies if it touches traps

        for trap in trap\_group.all\_trap\_group:

            if isinstance(trap, trapClasses):

                for rect in trap.rect\_group:

                    if self.rect.colliderect(rect):

                        return PLAYERDEATH

        # Return None if the player hasn't died

        return None

It check if the player collide with the enemies or traps. If yes, it will kill the player, else nothing will change.

### Map

#### Parse Map

def parseMap(self: mario.Map, map: list, barrier\_list: list):

    self.dead\_zone.append((\*barrier\_list[0:2], barrier\_list[2]\*BLOCKSIZE[0], barrier\_list[3]\*BLOCKSIZE[1]))

    for r\_num, row in enumerate(map):

        for c\_num, col in enumerate(row):

            if col == letter1:

                ground = Tile1()

                self.tileGroup.add(ground)

            elif col == "F":

                self.finish\_point = helper.FinishPoint(r\_num, c\_num)

            elif col == "C":

                tile = helper.Check\_point(r\_num, c\_num)

                self.checkpoint\_group.append(tile)

It take in a list of list which is the tile map of the level. It will check each cell and create object according to the letter of the cell. It will also create a deadzone under the map to kill any character falling of the map

## Testing Data

|  |  |  |
| --- | --- | --- |
| Function | Scenario | Expected Output |
| Player movement | “A” pressed | Character move left |
| “D” pressed | Character move right |
| “W” pressed | Character jumps |
| “W” pressed x 2 | Character jumps twice |
| “W” pressed x 3 | Character jump twice and not jump and third press |
| Checkpoint | Character move over the checkpoint and dies | The player respawn at checkpoint |
| Restart | Press restart through in-game menu | Restart and the beginning of the level |
| Ranking board |  | Screen displaying top 3 results of each level |
| Death | Player HP reduced to zero | Character dies, print death screen and respawn at checkpoint |
| Collision with trap | Character collide with traps | Character dies and respawn at checkpoint |
| Collision with enemies | Character collide with enemies | Character dies and respawn at checkpoint |
| Falling off the map | Character fall off the map | Character dies and respawn |
| Win | Character collide with flag pole | Win screen and return to the main menu |
| Camera Movement | Character tries to move off of the camera | The camera follows the character |
| Shoot Bullet | Pressed space bar | Bullet shoot off of the character |
| Bullet collision with enemies | Bullet hit enemies | Enemies health reduce |
| Enemies death | Enemies HP reduce to zero | Enemies death |
| Bullet collision with tiles | Bullet hit tiles | Bullet dies |
| Traps activation | Character move to activation zone | Trap get activated |

# Developing the Coded Solution

## Iteration 1

In the first iteration, I will make the home screen first.

It will include the following:

* Buttons Linked to each level
* Button Linked to menu

Classes Included:

* Main
* Menu
* Level button
* Setting
* Ranking Screen

#### Main

class Main:

    size: tuple = SIZE

    screen: pygame.Surface = pygame.display.set\_mode(size)

    clock: pygame.time.Clock = pygame.time.Clock()

    all\_sprite\_list: list = []

    currentScreen: str = MENU

    menu: Menu = Menu()

    setting: Setting = Setting()

    all\_sprite\_list.extend([menu, setting])

    status: list = []

It consist of the Menu and Setting. It has a list of status list to keep track of the state of the game.

##### Main Loop

def gameplay(self):

        done = False

        while not done:

            # -- User input and controls

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    sys.exit()

                self.keyResponse(event)

            # Mouse pointer interaction

            click,\_,\_ = pygame.mouse.get\_pressed()

            # only runs if it is left-clicked

            if click == True and not self.mouseBuffer.flag:

                self.mouseBuffer.tFlag()

                mouse = pygame.mouse.get\_pos()

                self.mouseResponse(mouse)

            #--Game logic goes after this comment

            self.logic()

            # -- Screen background is BLACK

            self.drawScreen()

            self.clock.tick(60)

It is main loop of the game and handles the logic flow of the game.

1. Response to any keyboard button press.
2. Check if the mouse is left-clicked. If yes, it will pass the mouse coordinate into subfunctions to check if it collides with anything.
3. It will then run logic which contains the function that are expected to run in every loop.
4. Draw out every object

Repeat

#### keyResponse

def keyResponse(self,event: pygame.event):

    if event.type == pygame.KEYDOWN:

        if event.key == pygame.K\_1:

            self.status.append(create\_level\_status\_code(1))

        elif event.key == pygame.K\_2:

            self.status.append(create\_level\_status\_code(2))

It handles input from the keyboard and update the status based on the keys pressed.

#### mouseResponse

def mouseResponse(self, position: tuple, click):

        if self.currentScreen == MENU:

            self.status = self.menu.mouseInteraction(position, self.status, click)

        elif self.currentScreen == SETTINGSCREEN:

            self.status = self.setting.mouseInteraction(position, self.status, click)I

It handles inputs from the pointer and update the status based on the button pressed. It checks the current screen to determine which screen’s mouseInteraction is called.

#### readStatus

def readStatus(self):

        i = 0

        while i < len(self.status):

            stat = self.status[i]

            if stat == SCREENTOSETTING:

                self.setCurrentScreen(SETTINGSCREEN)

            del self.status[i]

It iterates over the status and performs actions based on the status code.

##### Draw Screen

def drawScreen(self):

        self.screen.fill(BROWN)

        if self.currentScreen == MENU:

            self.menu.drawScreen(self.screen)

        elif self.currentScreen == SETTINGSCREEN:

            self.setting.drawScreen(self.screen)

This function draw the corresponding objects according to the screen it is in.

#### Menu

Initialisation

class Menu():

    def \_\_init\_\_(self, screen):

        self.size = SIZE

        self.screen = screen

        self.menuSpriteGroup = pygame.sprite.Group()

        self.levelButtonGroup = pygame.sprite.Group()

        self.settingButton = SettingButton(\*SETTINGS\_BUTTON\_SIZE, (SIZE[0] - 132, 20))

        self.menuSpriteGroup.add(self.settingButton)

        # The starting coordinate of the first button

        startY = 120 + LEVEL\_IMAGE\_SIZE[1] + LEVEL\_IMAGE\_BUTTON\_PADDING

        startX = EDGE\_LEVEL\_IMAGE\_PADDING + (LEVEL\_IMAGE\_SIZE[0]-LEVEL\_BUTTON\_SIZE[0])/2

        # Create level button according to the preset paddings

        for lvlNum in range(NUM\_OF\_LEVELS):

            levelBut = LevelButton(lvlNum+1, f"Level {lvlNum + 1}",\*LEVEL\_BUTTON\_SIZE,(startX,startY), "images/tempLevelImg.png")

            self.levelButtonGroup.add(levelBut)

            self.menuSpriteGroup.add(levelBut)

            startX += 264

The menu class contains the setting button and the button for different levels.

In the for loop, new object of LevelButton is created and added to levelButtonGroup, which will contains all the button to play the game.

#### Button Class

class Button(pygame.sprite.Sprite, ABC):

    def \_\_init\_\_(self,width, height, position, imageName):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.image = pygame.transform.scale(pygame.image.load(imageName), (width, height))

        self.rect = self.image.get\_rect()

        self.rect.x = position[0]

        self.rect.y = position[1]

    @abstractmethod

    def mouseInteraction(self,position, status):

        if self.rect.collidepoint(position):

            pass

        return status

    def keyResponse(self,event,status):

        return status

    def draw(self,screen, cam\_pos = pygame.math.Vector2(0,0)):

        screen.blit(self.image, (self.rect.x,self.rect.y))

It is abstract class that act as a blueprint for the other subclass that will be used later on. The is class will create a button with an images as the button.

#### Level Button

class LevelButton(Button):

    def \_\_init\_\_(self,level, name, width, height, position, imageName):

        super().\_\_init\_\_(width, height, position, imageName)

        self.name = name

        self.level = level

    def mouseInteraction(self,position, status):

        if self.rect.collidepoint(position):

            print("Level button mouse collide")

            self.initialiseGame()

        return status

    def initialiseGame(self):

        game = Game(self.level)

        game.play()

It is the level button. When it is click, the main screen will launch the corresponding level. (but since not level is implemented, only printing of a statement will be expected)

#### Word Button class

class WordButton(ABC):

    def \_\_init\_\_(self,width, height, position, color, textColor, txt):

        super().\_\_init\_\_()

        self.rect = pygame.Rect(\*position, width, height)

        self.color = color

        font = pygame.font.Font("freesansbold.ttf", FONTSIZE)

        self.txt = font.render(txt, True, textColor)

        fontSize = self.txt.get\_size()

        self.txt\_pos = (position[0] + (width - fontSize[0])/2, position[1] + (height - fontSize[1])/2)

    @abstractmethod

    def mouseInteraction(self,position, status):

        return status

    def keyResponse(self,event,status):

        return status

    def draw(self, screen):

        pygame.draw.rect(screen, self.color, self.rect, 0)

        screen.blit(self.txt, self.txt\_pos)

It is also an abstract based class for later use. It will create a button with a string in the middle of the button.

#### Button Mouse Interaction (Setting Button, Level Button etc)

def mouseInteraction(self,position: tuple, status: list):

        if self.rect.collidepoint(position):

            status.extend(SCREENTOSETTING)

        return status

Position is the coordinate of the mouse pointer while the status is the list of action that’s need to run. If the coordinate of the mouse pointer collides with the button, it will add a relevant command into the status list.

Graphical user interface, application

Description automatically generated

This is the screen I currently have. I have used the Super Mario sign as a temporary placeholder for the level button.

### Iteration 1 Testing

|  |  |  |
| --- | --- | --- |
| Input | Expected output | Result |
|  | A menu is shown with 3 menu buttons and 1 setting Button | As expected |
| Left-click on each level button | Print level button mouse collided | The string is printed multiple time in one click |
| Left-click on the menu button | Blank menu appears at the middle of the screen | As expected |

### Feedback from end user

Buttons have record the same click for multiple times.

I will continue to develop the game. I will need to fix the button registrating too many times issue and add a static game for the levels.

#### Success Criteria

* Screen is 810 x 540 pixel

## Iteration 2

In this iteration, I will fix the button registering multiple times for one click issure. Also, I will implement the initialisation and basic functions of the game.

I will make the classes for the game, player and map which includes basic movements and tiles loading from a csv file.

#### Button

def mouseInteraction(self,position: tuple, status: list, click):

    if click and self.rect.collidepoint(position) and not self.clicked:

        status.extend([self.status\_code])

        self.clicked = True

    elif not click:

        self.clicked = False

    return status

I have changed the mouseInteraction method in the Button and WordButton class to the above. The click argument is a Boolean value indicating where the mouse is left-clicked. It will then check if the mouse collide with the button. After the button is pressed on, it will lock the button until the left click is release so that the button will not be activated multiple times for one click

def gameplay(self):

    done = False

    while not done:

        # -- User input and controls

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                sys.exit()

            self.keyResponse(event)

        click,\_,\_ = pygame.mouse.get\_pressed()

        mouse = pygame.mouse.get\_pos()

        self.mouseResponse(mouse,click)

        self.readStatus()

        #--Game logic goes after this comment

        self.logic()

        # -- Screen background is BLACK

        self.drawScreen()

        self.clock.tick(60)

    #Endwhile

In this new version, the mouse interaction methods will run no matter the mouse is left-clicked or not.

#### Game initialisation

f initialiseGame(self, level):

    game = Game(level)

    game.play()

It will initialise the game according to the level attribute of the button the user clicked on.

#### Game

def \_\_init\_\_(self, level = 1):

        self.size = SIZE

        self.screen = pygame.display.set\_mode(self.size)

        self.clock = pygame.time.Clock()

        self.map = Map("map" +str(level) +".csv")

        self.level = level

        self.gameSpriteGroup = []

        self.player = Player()

        self.gameSpriteGroup.extend([self.player, self.map])

It contains the map class and a player class.

#### Player

def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load("images/redRect.png"), PLAYER\_SIZE)

        self.rect = self.image.get\_rect()

        self.rect.x = PLAYERSTARTPOS[0]

        self.rect.y = PLAYERSTARTPOS[1]

        self.xSpeed = 0

        self.ySpeed = 0

It contain a rectangle object (provide hitbox) and a image with its basic coordinates and speeds.

#### Map

def \_\_init\_\_(self, filename):

        temp = []

        with open(filename, 'r') as f:

            reader =  csv.reader(f)

            next(reader)

            for row in reader:

                temp.append(tuple(row))

        self.groundSpriteGroup = pygame.sprite.Group()

        self.tileGroup = pygame.sprite.Group()

        self.parseMap(temp)

It takes in a file name as argument to open a csv file which contains the data about the map, The data is them passed to parseMap method to initialise the map of the level. The groupSpriteGroup will contain all ground tiles while the tileGroup will contains every tiles

#### parseMap

def parseMap(self,tiles):

    for row in tiles:

        # Get the relative position and change it into pygame coordinates

        relPos = (int(row[1]), int(row[2]))

        dePos = relativeCoor2DeCoor(relPos)

        # The number of blocks extand in the x and y directions

        width, height = int(row[3]), int(row[4])

        # Check for the type of block

        if row[0] == "ground":

            for i in range(width):

                for j in range(height):

                    groundTile = Ground((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

                    self.groundSpriteGroup.add(groundTile)

                    self.tileGroup.add(groundTile)

        elif row[0] == "airTile":

            for i in range(width):

                for j in range(height):

                    airTile = AirTile((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

                    self.tileGroup.add(airTile)

It loop through the list from the csv file and create objects according to their coordinates and types.

1. It changes the coordinate from coordinates from in terms of how many blocks into the actual coordinate in the Pygame. (relativeCoor2DeCoor())
2. The height and width is the number of consecutive blocks in the x and y axis.
3. It will checks the type of blocks and create different object according to the height and width.

#### Tiles

class Tile(pygame.sprite.Sprite):

    def \_\_init\_\_(self,position, imgFileName):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load(imgFileName),BLOCKSIZE)

        self.rect = self.image.get\_rect()

        self.rect.x, self.rect.y = position

class Ground(Tile):

    def \_\_init\_\_(self,position):

        super().\_\_init\_\_(position,"images/groundTile.png")

class AirTile(Tile):

    def \_\_init\_\_(self,position):

        super().\_\_init\_\_(position, "images/airTile.png")

These classed are compositions of the map. Tile class is the abstract parent class which create a rectangle sprite with the image of the tile and position. The children class Ground and AirTile are specific types of Tile with specific images.

Chart

Description automatically generated with medium confidence

The red rectangle is the player with a placeholder images. The map can be load from a flat file.

### Iteration 2 Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Output | Solution |
| Click on each level buttons | Screen change to the corresponding level | As expected |  |
|  | Map loaded | As expected |  |
|  | character loaded | As expected |  |

### Feedback from end user

The graphics looks descent but no actual game yet. The character will go out of the screen.

#### Success Criteria

* Each block is 30 x 30 pixel in size
* Button for user to click on to access the game levels

## Iteration 3

In this iteration, I will add some basic interaction in the game, including the basic left, right, jump movement of the character and a camera that will follows the character when it is about the leave the screen.

For the user interface, I have added an in-game menu so that the player can exit the game.

#### Camera Class

class Camera:

    def \_\_init\_\_(self,player, vertices: list):

        # The edges of the camera

        self.vertices = vertices

        self.player = player

        # The position of the camera

        self.position = vec(0,0)

        self.screenW, self.screenH = SIZE

        self.left\_padding = PLAYER\_LEFT\_PADDING

        self.right\_padding = PLAYER\_RIGHT\_PADDING

        self.position.x = self.player.rect.x - self.left\_padding

        self.position.x = max(self.vertices[0], self.position.x)

        self.position.x = min(self.vertices[1] - self.screenW, self.position.x)

        self.position.x = int(self.position.x)

    def scroll(self):

        # If the player coordinate is smaller than the left padding relative to the camera

        if self.player.rect.x < self.position.x + self.left\_padding:

            # Set the position fo the camera so that the player is on the left padding unless the camera reaches the edge of the level

            self.position.x = self.player.rect.x - self.left\_padding

            self.position.x = max(self.vertices[0], self.position.x)

        # If the player coordinate is larger than the right padding relative to the camera

        elif self.player.rect.x > self.position.x + self.screenW - self.right\_padding:

            # Set the position fo the camera so that the player is on the right padding unless the camera reaches the edge of the level

            self.position.x = self.player.rect.x + self.right\_padding - self.screenW

            self.position.x = min(self.vertices[1] - self.screenW, self.position.x)

The camera is used to control the view of the game. The vertices attribute are the edges of the camera view. Left and right padding are the padding that the player should have to the left and right of the screen

The camera class contain a reference to the player object as well as the set boundary of the the level. It will refer the player position. Make the camera position (top right of the camera) so that the character doesn’t not exceed the padding in the screen.

The scroll method adjust the position of the camera as the game plays. If the player is exceeding the left padding, the camera will move left until it reaches the edge of the level. If the player is exceeding the right padding, the camera will move right until it reaches the edge of the level.

#### Game

class Game:

    def \_\_init\_\_(self, level = 1):

        self.size = SIZE

        self.screen = pygame.display.set\_mode(self.size)

        self.clock = pygame.time.Clock()

        self.map = Map("map" +str(level) +".csv")

        self.level = level

        self.menu\_button = MenuButton()

        self.status = []

        self.openMenu = False

        self.menuScreen = GameMenuScreen()

        self.done = False

        self.mouseBuffer = MouseBuffer()

        self.gameSpriteGroup = []

        self.player = Player(HORIZONAL\_MAX\_SPEED, self.map.vertices)

        self.camera = Camera(self.player, self.map.vertices)

        self.gameSpriteGroup.extend([self.player, self.map, self.menu\_button])

I have added the menu button, game menu, and camera class into the game class.

#### Game.keyresponse

def keyResponse(self, event):

    if self.openMenu:

        self.status = self.menuScreen.keyResponse(event,self.status)

    else:

        self.status = self.player.keyResponse(event, self.status)

It will call menuScreen or player keyResponse method determined by if the menu is opened or not.

#### Game.mouseResponse

def mouseResponse(self, position):

    self.status = self.menu\_button.mouseInteraction(position, self.status)

    if self.openMenu:

        self.status = self.menuScreen.mouseInteraction(position, self.status)

It will check if the menu button is clicked and only check the object in the menu if the menu is opened

#### Player.keyResponse

def keyResponse(self,event, status):

        if event.type == pygame.KEYDOWN:

            # Move left

            if event.key == pygame.K\_a:

                self.set\_speed\_x(-self.horizonal\_max\_speed)

            # Move right

            elif event.key == pygame.K\_d:

                self.set\_speed\_x(self.horizonal\_max\_speed)

            #jump

            elif event.key == pygame.K\_w:

                self.jump()

            # Shoot (empty)

            elif event.key == pygame.K\_SPACE:

                self.start\_shooting()

        elif event.type == pygame.KEYUP:

            if event.key == pygame.K\_a or event.key == pygame.K\_d:

                self.set\_speed\_x(0)

            elif event.key == pygame.K\_SPACE:

                self.stop\_shooting()

        return status

I have added keyresponse so that the user can use WASD to control the character.

When the A or D key is pressed down, the horizontal speed of the character will be change to moving left or right. If the space is pressed, the player will jump. When the A or D key is released, the speed will be set back to zero.

For the space key, it is planned to used for shooting but it has not been implemented yet.

#### Player.update

def update(self):

        # Change the x coordinate by horizontal speed

        self.rect.x += self.xSpeed

        # Change the x coordinate so that it doesn't exceed the boarder

        self.rect.x = min(self.rect.x, self.right\_border)

        self.rect.x = max(self.rect.x, self.left\_border)

        # Change the y coordinate by vertical speed

        self.rect.y += self.ySpeed

        # Acceleration by "gravity"

        self.ySpeed += GRAVITY

        # A temporary ground for the player so that it does not falls to nowhere

        # Require change to tile based

        self.rect.y = min(self.rect.y, 180)

I added basic movements to the player. It will prevent the character from moving outside the boarder of the level. I have added a temporary “ground” for the character so that it does not fall to nowhere.

Chart, histogram

Description automatically generated

#### Close Game Button

class QuitButton(WordButton):

    def \_\_init\_\_(self, position: tuple):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Exit Game", CLOSEGAME)

It is a subclass of WordButton that would add Close game status when it is clicked.

#### Read Status

def readStatus(self):

    """

    Read the status code of the game

    """

    for stat in self.status:

        # Turn screen mode to game menu

        if stat == SCREENTOGAMEMENU:

            self.currentScreen = GAMEMENU

        # Exit the current game

        elif stat == CLOSEGAME:

            self.done = True

New read status including the close game

#### Menu

Graphical user interface

Description automatically generated with medium confidence

It is the in-game menu where you can exit the game or get the control menu (not yet implemented)

### Iteratioh 3 Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Solution |
| “A” pressed | Character move left | As expected |  |
| “D” pressed | Character move right | As expected |  |
| “W” pressed | Character jumps | It can jump but it doesn’t collide with the tiles | Make all collision relative to the tile |
| Click on the in-game menu button | In-game menu popup | As expected |  |
| Click on exit game button | Exit game | As expected |  |

### Feedback for iteration 3

* The character does not collide with the tiles
* The words in buttons are badly align
* The character is a bit too big for a small map

#### Success Criteria

* The user can use a keyboard as input to control the character
* The character can move left and right
* The camera will follow the character whenever it reaches the edges of the screen

## Iteration 4

In this iteration, I will add a death screen, restart option and control menu to the game. In addition, I also change the movement of the player so that it will interact the tiles and traps for later development.

#### Death Zone

        elif row[0] == "barrier":

            self.dead\_zone.append((\*dePos, width\*BLOCKSIZE[0], height\*BLOCKSIZE[1]))

I have added a type of invisible tile which essentially act as a death zone where the character will die if it touches it. It is put below the map so that the character will die after they fall off the map.

##### Player.check\_dead\_zone

def check\_dead\_zone(self,dead\_zone):

        for dead in dead\_zone:

            if self.rect.x >= dead[0] and self.rect.x <= dead[0] + dead[2] and self.rect.y >= dead[1] and self.rect.y <= dead[1] + dead[3]:

                self.health -= 1000

def checkdeath(self):

        if self.health <= 0:

            return PLAYERDEATH

It checks if the player coordinates collide with the dead zone. If true, the player health will be reduced by 1000 (instant kill). If the player has less than or equal to 0 health left, it will return a status code indicating the player is dead and restart the game.

The checkdeath method kills the player if the health is lower or equal to 0. It will add the playerdeath status code to the status to finish the game.

#### Player movement

def update(self, tiles):

    self.movementX()

    collided\_tiles = pygame.sprite.spritecollide(self, tiles, False)

    if len(collided\_tiles) != 0:

        self.collisionX(collided\_tiles)

    self.movementY()

    collided\_tiles = pygame.sprite.spritecollide(self, tiles, False)

    if len(collided\_tiles) != 0:

        self.collisionY(collided\_tiles)

1. Change the x coordinate of the player
2. Collision detection in x direction (pass in the tiles the character collides with)
3. Change the y coordinates of the player
4. Collision detection in y direction (pass in the tiles the character collides with)

##### Player.movementX

def movementX(self):

        self.rect.x += self.xSpeed

        # boarder collision

        self.rect.x = min(self.rect.x, self.right\_border)

        self.rect.x = max(self.rect.x, self.left\_border)

It changes the x coordinate according to the horizontal speed and stop the character if it goes pass the barrier of the level.

##### Player.collisionX

    def collisionX(self, tiles: list):

        """

        Horizonal interaction with tiles it collided with

        Args:

            tiles (list): The ground tiles

        """

        # For moving right

        if self.xSpeed > 0:

            # If there is collision, change x coordinate to the left to the block it collide with

            for tile in tiles:

                self.rect.x = min(tile.rect.x - PLAYER\_SIZE[0], self.rect.x)

        # For moving left

        elif self.xSpeed < 0:

            # IF there are collisions, change x coordinate to the right of the block it collide with

            for tile in tiles:

                self.rect.x = max(tile.rect.right, self.rect.x)

This method handles the horizontal collision of the character. It determine the direction the character is moving by the speed of the character.

If the player is moving right (speed > 0), it will change the x coordinate to the left side of the most left tile it collides with so that the character is side to side to the block while not overlapping.

If the player is moving left (speed < 0), it will change the x coordinate to the right side of the most right tile it collides with so that the character is side to side to the block while not overlapping.

##### Player.movementY

def movementY(self):

    """

    Vertical movement logic

    """

    self.rect.y += self.ySpeed

    self.ySpeed += GRAVITY

It moves the character according to the vertical speed and add a constant to the speed to mimic the gravitational acceleration.

##### Player.collisionY

def collisionY(self, tiles: list):

    """

    Veritical interaction with tiles it collided with

    Args:

        tiles (list): Ground tiles

    """

    # Moving down

    if self.ySpeed > 0:

        self.on\_ground = True

        # Set the y coordinate to the top of the collide blocks

        for tile in tiles:

            self.rect.y = min(tile.rect.top - PLAYER\_SIZE[1], self.rect.y)

    # Moving up

    elif self.ySpeed < 0:

        # Set the y coordinate to the top of the collide blocks

        for tile in tiles:

            self.rect.y = max(tile.rect.bottom, self.rect.y)

    # Taking away any vertical velocity if collision

    self.ySpeed = 0

This a method of handling vertical collision between the character and the tiles. It checks the player’s vertical speed to determine if it is moving up or down.

If the character is moving down, it will set the on\_ground to be true. (Since this function only runs if it collides with any tile. This is for the character to reset the jumping number so that it can jump again) It will set the y coordinates of the character to the top of the highest tile it collides with.

If the character is moving up, it will set the y coordinate to the bottom of the lowest tiles it collides with and thus make the character just stick to the bottom without any overlapping.

It will also set the vertical speed to zero. It is to prevent the speed by gravitational acceleration builds up so that the vertical displacement in one frame exceed the floor and thus fall through the floor.

#### Death Screen

class DeathScreen():

    def \_\_init\_\_(self):

        font = pygame.font.Font("freesansbold.ttf", 100)

        self.txt = font.render("Defeated", True, DARKBLUE)

        self.txt\_pos = (182, 130)

    def draw(self,screen):

        screen.blit(self.txt, self.txt\_pos)

The death screen will only have text overlaying the stopped game. Although there is only one attribute in this class, I will still create a class for it for better modular program and easier to add more features to the death screen if needed.

A picture containing shape

Description automatically generated

#### Game Menu Screen

class GameMenuScreen():

    def \_\_init\_\_(self):

        self.background = Background(706, 381, (47, 48), YELLOW)

        self.closeButton = CloseButton(24,24, (741, 36), RETURNTOGAME)

        self.quitButton = QuitButton((331, 306))

        self.controlButton = InstructionButton((331,118))

        self.restartButton = RestartButton((331, 212))

        self.gameMenu\_sprite\_group = [self.background, self.closeButton, self.quitButton,self.controlButton, self.controlButton, self.restartButton]

It contains:

* Background
  + Just a coloured background.
* Close button
  + To close the game menu
* Quit button
  + To exit the game and back to the home screen
* Control button
  + To access the control menu
* Restart button
  + To restart the game at the starting point

A picture containing website

Description automatically generated

#### Close button class

class CloseButton(Button):

    def \_\_init\_\_(self,width, height, position, statusCode):

        super().\_\_init\_\_(width, height, position, "images/closeButton.jpg")

        self.statusCode = statusCode

    def mouseInteraction(self, position, status):

        if self.rect.collidepoint(position):

            status.extend([self.statusCode])

        return status

It is just a rectangle that when it is clicked, it will send a status to the game so that the menu is closed.

#### Quit Button

class QuitButton(WordButton):

    def \_\_init\_\_(self, position):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Exit Game")

    def mouseInteraction(self, position, status):

        if self.rect.collidepoint(position):

            status.extend([CLOSEGAME])

        return status

It is a button where when it is clicked it will send a status to the game so that it is exit back to the main menu.

#### Instruction button

class InstructionButton(WordButton):

    def \_\_init\_\_(self, position):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Instructions")

    def mouseInteraction(self, position, status):

        if self.rect.collidepoint(position):

            print("Instruction clicked")

            status.append(SCREENTOINSTRUCTION)

        return status

It is a button that when it is clicked, it will send status code to the game so that the instruction menu is shown.

#### Restart button

class RestartButton(WordButton):

    def \_\_init\_\_(self, position):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Restart")

    def mouseInteraction(self, position, status):

        if self.rect.collidepoint(position):

            status.extend([RESTARTGAME])

        return status

It is the same as other word button but it will send status that will restart the game.

#### Instruction screen class

class InstructionScreen():

    def \_\_init\_\_(self, closeStatusCode: str):

        # Instructions

        self.instructions = [

            " 'A' key for going to the left ",

            " 'D' key for going to the right",

            " 'W' key for jumping",

            " 'SPACE' key for shooting"

        ]

        self.fonts = []

        # for generation class

        font = pygame.font.Font("freesansbold.ttf", FONTSIZE)

        # Generate surface object for the instuctions

        start\_y = 108

        for instruction in self.instructions:

            txt = font.render(instruction, True, BLACK)

            fontSize = txt.get\_size()

            # It is to center the text

            txt\_pos = (76 + (648 - fontSize[0])/2, start\_y + INSTRUCTION\_MENU\_PADDING)

            # Add the padding for the coordinate of the next text

            start\_y += fontSize[1] + INSTRUCTION\_MENU\_PADDING \* 2

            self.fonts.append((txt,txt\_pos))

        self.background = Background(648,336,(76, 64), SETTINGSCREENCOLOR)

        self.closeButton = CloseButton(24,24, (712, 52), closeStatusCode)

        self.instruction\_sprite\_group = [self.background, self.closeButton]

This class responsible for creating the instruction screen. It contains fixed instruction and create a font class using freesansbold.ttf font. Using the font class and the list of string, it create a list of surface object for the list of instruction.

Graphical user interface, text, application

Description automatically generated

### Iteration 4 Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Solution |
|  | Player doesn’t fall off the floor | As expected |  |
| Move right and collide | Player stop when colliding | As expected |  |
| Move left and collide | Player stop when colliding | As expected |  |
| Move off an edge | Fall off | As expected |  |
| Press “W” twice | Double jump | As expected |  |
| Press “W” twice after falling off an edge | Double jump | As expected |  |
| Fall off the floor | Player dies and present death screen | As expected |  |
| Press menu button | Menu pop up | As expected |  |
| Press instruction button inside the in-game menu | Instructions menu pop up | As expected |  |
| Press restart button | Game restart | As expected |  |
| Press exit game button | Exit to the main menu | As expected |  |

### User feedback

The basic movement of the game is implemented. Need to start making enemies and traps

#### Success criteria

* There will be control menu showing the controls of the game
* Death screen
* The character can move left and right
* The character can jump and double jump
* The character will die if they fall off the map

## Iteration 5

In this iteration, I will make different traps and enemies object. In addition, I will make checkpoint for respawn and flag to finish the game.

#### Trap Group

class TrapGroup():

    def \_\_init\_\_(self, filename: str):

        self.all\_trap\_group = []

        self.parseFile(filename)

    def parseFile(self,filename: str):

        # Maybe not use standard csv file but determine how to read the content by the first column

        with open(filename, 'r') as f:

            reader =  csv.reader(f)

            next(reader)

            for row in reader:

                relPos = (int(row[1]), int(row[2]))

                dePos = relativeCoor2DeCoor(relPos)

                if row[0] == "normal\_spike":

                    for i in range(int(row[5])):

                        for j in range(int(row[6])):

                            trap = Spike(relativeCoor2DeCoor((int(row[1])+ i, int(row[2])+j)), int(row[3]), int(row[4]))

                            self.all\_trap\_group.append(trap)

                elif row[0] == "up\_spike":

                    trap = SpikeUp(dePos, int(row[3]), int(row[4]),apply(row[5:9], int), int(row[11]), int(row[12]))

                    self.all\_trap\_group.append(trap)

                elif row[0] == "grow\_spike":

                    trap = GrowSpike(dePos, int(row[3]), int(row[4]),apply(row[5:9], int), int(row[11]), int(row[12]))

                    self.all\_trap\_group.append(trap)

                elif row[0] == "hori\_spike":

                    trap = HorizontalSpike(dePos, int(row[3]), int(row[4]))

                    self.all\_trap\_group.append(trap)

                elif row[0] == "hori\_move\_spike":

                    trap = MoveableHoriSpike(dePos, int(row[3]), int(row[4]),apply(row[5:9], int), int(row[11]), int(row[12]))

                    self.all\_trap\_group.append(trap)

It is used load the csv file containing data of the traps. It iterates through each row of the csv file and create the corresponding traps. The information in the csv file contains the type of traps, the coordinate of traps, the size, the activation zone and the movement of the trap (if there is one). In addition, it also contains the number of repetition of the same trap in the horizontal direction and the vertical direction. It will append all traps into the all\_trap\_group which contains everytrap inside the level.

#### Spike

class Spike:

    def \_\_init\_\_(self, position: tuple, base: int, height: int):

        # Position of the square the spike is int

        self.x, self.y = position[0], position[1]

        # The base and height of the triangle

        self.base = base

        self.height = height

        self.rect\_group = []

        self.createRect()

    # The spike will made of 19 rectangle stacking on top of each other, mimicing a isosceles triangle. (The 20th has width of 0, so it will be ignore)

    def createRect(self):

        # The decrease in base for every rectangle on top

        base\_change = self.base / 40

        height\_change = self.height / 20

        # Base of the first triangle

        base\_length = self.base - base\_change\*2

        # The coordinate of the first rectangle in the first iteration

        left = self.x + base\_change

        top = self.y + self.height-height\_change

        for \_ in range(19):

            rect = pygame.Rect(left, top, base\_length, height\_change)

            self.rect\_group.append(rect)

            base\_length -= base\_change\*2

            left += base\_change

            top -= height\_change

Shape

Description automatically generated with medium confidenceSince the collision detection function in Pygame only allows point and rectangle objects, I cannot use a triangle surface and image. The spike is created by making 19 rectangles stacking on top of each other, mimicking an isosceles triangle. (the 20th triangle will have width of0, so it is ignored)

The program will create rectangles one on top of each other so that it looks like a small isosceles triangle.

(The image on the right is a spike when it is magnified)

#### Activate class

class ActivateObjects:

    def \_\_init\_\_(self,zone: list):

        self.zone = pygame.Rect(\*zone)

        self.activate = False

    def detect(self, player\_rect: pygame.Rect):

        if player\_rect.colliderect(self.zone):

            self.activate = True

It an abstract class that act for any object that can be activated by the player’s position

#### Spike Up class

class SpikeUp(Spike, ActivateObjects):

    def \_\_init\_\_(self, position: tuple, base: int, height: int, zone: list, up: int, hori\_dir = 0): #zone = (leftx, rightx, width, height)

        Spike.\_\_init\_\_(self, (position[0], position[1]), base, height)

        ActivateObjects.\_\_init\_\_(self,zone)

        self.tar\_y = position[1] - up\*BLOCKSIZE[1]

        # Number of block upward it should moves

        self.vert\_dir = up

        # Number of block to the right it should moves

        self.hori\_dir = hori\_dir

        self.tar\_x = position[0] + hori\_dir \* BLOCKSIZE[0]

    # Return if the player collides with the spike

    def player\_interaction(self, player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

        if self.activate and player\_rect.collidelist(self.rect\_group) != -1:

            return True

        return False

It is a child class of Spike and Activate object where it inherit the rectangle and activation zone attribute as well as detect method from the parent classes.

It has the attribute for the number of blocks it should moves up and right

The player interaction function will detect whether the player is in activation zone if it is activated. If it is not activated, it will check if the player collides with any of the rectangles in the spike. It will return true if the player collides.

#### SpikeUp.logic

# Move the spike if it is activated and it has not reached its final position

    def logic(self):

        if self.activate:

            # Move up

            if self.y > self.tar\_y and self.vert\_dir > 0:

                self.y -= UP\_SPEED

                for rect in self.rect\_group:

                    rect.y -= UP\_SPEED

            # Move down

            elif self.y < self.tar\_y and self.vert\_dir < 0:

                self.y += UP\_SPEED

                for rect in self.rect\_group:

                    rect.y += UP\_SPEED

            # Move right

            if self.x < self.tar\_x and self.hori\_dir > 0:

                self.x += HORI\_SPEED

                for rect in self.rect\_group:

                    rect.x += HORI\_SPEED

            # Move left

            elif self.x > self.tar\_x and self.hori\_dir < 0:

                self.x -= HORI\_SPEED

                for rect in self.rect\_group:

                    rect.x -= HORI\_SPEED

It checks if the coordinate is in the desire position yet and move it to that direction if needed. For example, if it should be moving up and the y coordinate is still smaller than the target y coordinate, it will move every rectangle in the spike up by a constant value. It checks for all directions.

It is the activation zone of a Spike up block: (The yellow rectangle is the activation zone of a Spike)A picture containing shape

Description automatically generated

#### Appear Block class

class Appear\_block(ActivateObjects, Ground):

    def \_\_init\_\_(self,position: tuple):

        ActivateObjects.\_\_init\_\_(self, (position[0]-10, position[1]-10, BLOCKSIZE[0]+20, BLOCKSIZE[1]+20))

        Ground.\_\_init\_\_(self,position)

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        if self.activate:

            screen.blit(self.image, (self.rect.x - cam\_pos.x, self.rect.y))

    def player\_interaction(self,player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

It is a child class that inherits form ActivateObjects and Gound tile classes. I will create a activation zone 10 pixel around the actual block so that the user can have some reaction time before it hits the block. The block will only be drawn if the block is activated.

The player interaction method will try to detect the player position and see if it collide with the activation zone. If yes, it will activate the block

#### Disappear block class

class DisappearBlock(ActivateObjects, Ground):

    def \_\_init\_\_(self,position: tuple):

        # easier

        # ActivateObjects.\_\_init\_\_(self, (position[0]- 3, position[1]- 3, BLOCKSIZE[0] + 6, BLOCKSIZE[1] + 6))

        ActivateObjects.\_\_init\_\_(self, (position[0], position[1], BLOCKSIZE[0], BLOCKSIZE[1]))

        Ground.\_\_init\_\_(self,position)

        self.rect = NULL

        self.position = position

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        if not self.activate:

            screen.blit(self.image, (self.position[0] - cam\_pos.x, self.position[1]))

    def player\_interaction(self,player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

It is the same as the appear block. However, instead of only drawing it after it is activated. It will disappear once the player has touched it.

The following code is added to the Map.parseMap method to incorporate these two tiles in the map.

      # Hidden block (only appear if the player is close)

        elif row[0] == "appear\_block":

            for i in range(width):

                for j in range(height):

                    tile = Appear\_block((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

                    self.tileGroup.add(tile)

        elif row[0] == "disappear\_block":

            for i in range(int(row[3])):

                for j in range(int(row[4])):

                    trap = DisappearBlock((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

                    self.disappear\_tiles.append(trap)

The disappear tiles will not be append to the tile group since it is not a real tile (it will never interact with the player).

#### Grow Spike

class GrowSpike(Spike, ActivateObjects):

    def \_\_init\_\_(self, position: tuple, base: int, height: int, zone: list, up: int, hori: int):

        Spike.\_\_init\_\_(self, position, base, height)

        ActivateObjects.\_\_init\_\_(self,zone)

        self.tar\_height = height\*up

        self.tar\_base = hori \* base

It is a subclass of Activate object and Spike. In addition to the parent attributes, I have added the target height and base to indicate how the final spike should be.

def logic(self):

        flag = False

        if self.activate and self.height < self.tar\_height:

            self.height += GROW\_SPEED

            self.y -= GROW\_SPEED

            flag = True

        if self.activate and self.base < self.tar\_base:

            self.base += GROW\_SPEED

            self.x -= GROW\_SPEED /2

            flag = True

        if flag:

            self.rect\_group = []

            base\_change = self.base / 40

            height\_change = self.height / 20

            base\_length = self.base - base\_change\*2

            left = self.x + base\_change

            top = self.y + self.height-height\_change

            for i in range(19):

                rect = pygame.Rect(left, top, base\_length, height\_change)

                self.rect\_group.append(rect)

                base\_length -= base\_change\*2

                left += base\_change

                top -= height\_change

This function will ‘grow’ size of the spike. It will first check if the spike is activated. If the spike is activated and it height or base is smaller than the target, it will remove all current rectanges and create a updated version of rectangles group with the new position, height and base.

class HorizontalSpike():

    def \_\_init\_\_(self, position: tuple, base: int, height: int):

        self.x = position[0]

        self.y = position[1]

        self.base = base

        self.height = height

        self.rect\_group = []

        self.createRect()

    def createRect(self):

        base\_change = self.base / 40

        height\_change = self.height / 20

        base\_length = self.base - base\_change\*2

        left = self.x

        top = self.y + height\_change

        for i in range(19):

            rect = pygame.Rect(left, top, height\_change, base\_length)

            self.rect\_group.append(rect)

            base\_length -= base\_change\*2

            left += height\_change

            top += base\_change

Shape

Description automatically generated with medium confidenceThis is the horizontal spike where the tip of the spike will face the right. It works the same as the vertical facing spike.

#### Moving Horizontal Spike

class MoveableHoriSpike(HorizontalSpike, SpikeUp):

    def \_\_init\_\_(self, position: tuple, base: int, height: int, zone: list, up: int, hori\_dir = 0):

        SpikeUp.\_\_init\_\_(self, position, base, height, zone, up, hori\_dir)

        HorizontalSpike.\_\_init\_\_(self,position, base, height)

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        #pygame.draw.rect(screen, YELLOW, pygame.Rect(self.zone.left - cam\_position.x, self.zone.top-cam\_position.y, self.zone.width, self.zone.height))

        HorizontalSpike.draw(self,screen, cam\_position)

    def player\_interaction(self,player\_rect: pygame.Rect):

        SpikeUp.player\_interaction(self,player\_rect)

    def logic(self):

        SpikeUp.logic(self)

It is a composition class of Spike up and horizontal. While the logics is the same as the spike up, the graphics and orientation of the rectangle are inherited from horizontal spike class.

#### Checkpoint

class Check\_point(Tile, ActivateObjects):

    def \_\_init\_\_(self, position: tuple):

        Tile.\_\_init\_\_(self, position, os.path.join("images","respawn\_before.png"))

        ActivateObjects.\_\_init\_\_(self, (position[0]-10, position[1]-10, BLOCKSIZE[0] + 20, BLOCKSIZE[1] + 20))

    def player\_interaction(self,player\_rect: pygame.Rect):

        if player\_rect.colliderect(self.zone):

            self.image = pygame.transform.scale(pygame.image.load(os.path.join("images","respawn.png")), BLOCKSIZE)

            return [self.x, self.y +18]

It is a check point block. It will change its color whenever the character collide with it and return a checkpoint coordinate to the main game. It will set a new checkpoint for the player so that when the player dies, it will respawn at that point instead of restarting from the start.

#### Finish Flag

class FinishPoint(Tile):

    def \_\_init\_\_(self, position: tuple):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images", "flag\_red.png")), (BLOCKSIZE[0], BLOCKSIZE[1]\*5))

        self.rect = self.image.get\_rect()

        self.rect.x = position[0]

        self.rect.y = position[1]

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        screen.blit(self.image, (self.rect.x - cam\_pos.x, self.rect.y))

It is a finish flag. When the player touches the flag, it will activate the wining procedure, where the game will end and victory screen will be presented.

#### Goomba class

class Enemy(pygame.sprite.Sprite):

    def \_\_init\_\_(self, position: list, x\_boundary: int, y\_boundary: int, imgName: str, size: tuple, hp: int):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load(imgName), size)

        self.size = size

        self.rect = self.image.get\_rect()

        self.rect.x = position[0]

        self.rect.y = position[1]

        self.x\_boundary = x\_boundary

        self.y\_boundary = y\_boundary

        self.x\_speed = 1

        self.hp = hp

        self.max\_hp = self.hp

        self.h\_bar\_height = round(self.rect.height\*0.1)

    def change\_hp(self, value: int):

        """

        Change the HP of the enemy

        Args:

            value (int): the amount of change in HP

        """

        self.hp += value

    def update(self):

        """

        Update the position and speed of the enemy

        """

        self.rect.x += self.x\_speed

        if self.x\_speed >0 and self.rect.x > self.x\_boundary[1]-self.size[0]:

            self.x\_speed \*= -1

        elif self.x\_speed < 0 and self.rect.x < self.x\_boundary[0]:

            self.x\_speed \*= -1

    def draw(self, screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        """

        Draw the enemy on the screen

        Args:

            screen (pygame.Surface): pygame display surface

            cam\_position (pygame.math.Vector2): the position of the camera

        """

        screen.blit(self.image, (self.rect.x - cam\_position.x, self.rect.y))

        pygame.draw.rect(screen, RED, (self.rect.left - cam\_position.x, round(self.rect.bottom - self.rect.height\*0.1), self.rect.width, self.h\_bar\_height))

        pygame.draw.rect(screen, LIGHTGREEN, (self.rect.left - cam\_position.x, round(self.rect.bottom - self.rect.height\*0.1), (self.rect.width \* self.hp/self.max\_hp), self.h\_bar\_height))

class Goomba(Enemy):

    def \_\_init\_\_(self, position, x\_boundary, y\_boundary):

        super().\_\_init\_\_(position, x\_boundary, y\_boundary, os.path.join("images","goomba.png"), GOOMBA\_SIZE, 30)

The enemy class is a subclass of Pygame sprite with its image, position, x and y boundaries, horizontal speed and hitpoints. The Goomba is a specific type of enemy where it only moves in the horizontal direction in a predetermined area.

#### Player.enemyinteraction

def enemy\_interaction(self,enemy\_sprite\_group: list, trap\_group: TrapGroup):

        """

        Interaction between enemy, traps and player

        Args:

            enemy\_sprite\_group (list): Group of enemy sprite

            trap\_group (TrapGroup): Traps

        """

        # Player dies if it touches with any enemy

        for enemy in enemy\_sprite\_group:

            if type(enemy) == Goomba:

                if pygame.sprite.collide\_rect(self, enemy):

                    self.health -= 200

                    return

        # Player dies if it touches traps

        for trap in trap\_group.all\_trap\_group:

            if type(trap) in [Spike, SpikeUp,GrowSpike, HorizontalSpike,MoveableHoriSpike]:

                for rect in trap.rect\_group:

                    if self.rect.colliderect(rect):

                        self.health -= 200

                        return

This the enemy interaction function within the player update. It check if it collides with any of enemy. If it collides with any of the traps or enemies, it will reduce the health of the player by 200, essentially killing the player.

### Iteration 5 testing

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected output | Actual output | Solution |
| Load trap csv | All trap presented on the screen | As expected |  |
| Player touches Spike | Player dies | As expected |  |
| Player collide with spike up activation zone | Spike up moves as designed | As expected |  |
| Player collide with spike up | Player dies | As expected |  |
| Player collide with appear block | Block appears | As expected |  |
| Player Collide with disappear block | Block disappear and no collision | As expected |  |
| Player collide with grow spike activation zone | Spike change in size | As expected |  |
| Player collide with activated grow spike | Player dies | As expected |  |
| Player collide with horizontal spike | Player dies | As expected |  |
| Player collide with moving horizontal spike activation zone | Spike moves | As expected |  |
| Player collide with finish flag | Game ends | As expected |  |
| Player collide with checkpoint and dies | Player respawn in the checkpoint after a few seconds | As expected |  |
|  | Goomba moves within the designated area on its own | As expected |  |
| Player collide with Goomba | Player dies | As expected |  |

#### User Feedback

The main design of the game is pretty good.

Things to add:

* Screen after winning
* Database to store all successful entries

#### Success Criteria

* Infinite lives
* The character respawns at the checkpoint after it dies
* There will be different traps and enemies
* More traps
* Whenever the character dies, the death count increase by one
* Flat file to store the information of enemies and traps

## Iteration 6

In this iteration, I will create code so that the player can shoot horizontal bullets to kill enemies and the victory screen. In addition, I will create the database to store the player entries and ranking board.

#### Bullet class

class Bullet(pygame.sprite.Sprite):

    def \_\_init\_\_(self, direction: int, position: tuple):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images", "redRect.png")), BULLET\_SIZE)

        self.rect = self.image.get\_rect()

        self.direction = direction

        self.rect.x = position[0]

        self.rect.y = position[1]

    def update(self, tiles: list, x\_boundary):

        self.rect.x += self.direction \* BULLET\_SPEED

        if len(pygame.sprite.spritecollide(self, tiles, False)) != 0 or self.rect.right < x\_boundary[0] or self.rect.left > x\_boundary[1]:

            self.kill()

    def draw(self, screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        screen.blit(self.image, (self.rect.x - cam\_position.x, self.rect.y))

It is a subclass of Pygame sprite. It contains a rectangle, its direction and its initial position.

After the bullet is created, it will move in the initial direction in a constant speed until it reaches the edges of the level or it collide with any of the tiles.

#### Player.shoot\_bullet() + Player.shoot\_timer()

def shoot\_bullet(self):

    """

    Shoot bullet

    """

    # Check if the player is allow to shoot bullets

    if self.bullet\_available:

        bullet = Bullet(self.face\_direction, self.rect.center)

        self.bullet\_group.add(bullet)

        self.bullet\_available = False

# Timer for time between each bullet

def shoot\_timer(self):

    if not self.bullet\_available:

        self.bullet\_timer += 1

    if self.bullet\_timer >= BULLET\_TIMER:

        self.bullet\_available = True

        self.bullet\_timer = 0

They are methods of the player class. The shoot\_bullet method will be called when the space bar is pressed. It will create a bullet object into the direction the player is facing with the position of the player. To prevent users from spamming the shoot button, I have added a shoot\_timer where it will set shoot bullet to be unavailable until the player waited for certain amount of time.

#### Bullet enemy interaction

def bullet\_enemy\_interaction(self, enemies: list):

    for enemy in enemies:

        hits = pygame.sprite.spritecollide(enemy, self.bullet\_group, False)

        enemy.change\_hp(-len(hits) \* BULLET\_DAMAGE)

        for bu in hits:

            bu.kill()

It will iterate through all the enemies. It will reduce the health of the enemy by the number of bullet it hits multiplying the bullet damage. The bullet will be killed after hitting any enemy.

#### Victory Screen

class WinText():

    def \_\_init\_\_(self):

        big\_font = pygame.font.Font("freesansbold.ttf",100)

        self.txt = big\_font.render("Victory", True, DARKBLUE)

        self.txt\_pos = (182, 130)

    def set\_time\_death(self, time, death):

        small\_font = pygame.font.Font("freesansbold.ttf",40)

        self.time\_txt = small\_font.render("Time used: "+str(time) + "  Death count: "+ str(death), True, DARKBLUE)

        self.time\_pos = (182+ (self.txt.get\_size()[0] - self.time\_txt.get\_size()[0])/2, 130 + self.txt.get\_size()[1])

    def draw(self,screen: pygame.Surface):

        screen.blit(self.txt, self.txt\_pos)

        screen.blit(self.time\_txt, self.time\_pos)

I will display the player has won with the time take and the death number. The game will automatically restart after a few seconds

A picture containing company name

Description automatically generated

#### Creating the SQLite database

CREATE TABLE Entries (id INTEGER PRIMARY KEY name VARCHAR(30) time INTEGER DEATHS INTEGER level INTEGER);

I have created a table to record the data of the entries of the player. Since there is no record and logging of each players detail, I will only have one table which is the table for all entries. It has primary key of id, name of varchar type (with maximum 30 characters), time of integer type, deaths of integer type and level of integer type.

#### Importing the database

import sqlite3

con = sqlite3.connect("data.db", check\_same\_thread=False)

db = con.cursor()

It create a connection to the database.

#### Committing the insert transaction

db.execute(f"INSERT INTO Entries (name, time, deaths, level) VALUES ('{self.player\_name}', {time\_count}, {death\_count}, {level});")

            con.commit()

It first create the transaction consist of a insert statement and commit to the actual database.

#### Ranking board

class RankingScreen:

    def \_\_init\_\_(self, closeStatusCode: str):

        self.font = pygame.font.Font("freesansbold.ttf", FONTSIZE)

        self.background = Background(648,336,(76, 64), SETTINGSCREENCOLOR)

        self.closeButton = CloseButton(24,24, (712, 52), closeStatusCode)

        self.instruction\_sprite\_group = [self.background, self.closeButton]

        self.background = pygame.Rect(134,114,532,237)

        self.refresh()

    def refresh(self):

        res1 = db.execute("SELECT Name, Time, Deaths, level FROM Entries WHERE level = 1 ORDER BY Time, Deaths ASC").fetchall()[0:3]

        res2 = db.execute("SELECT Name, Time, Deaths, level FROM Entries WHERE level = 2 ORDER BY Time, Deaths ASC").fetchall()[0:3]

        self.whitebox = []

        for i in range(7):

            self.whitebox.extend([pygame.Rect(139, 119 + i \* 33, 267, 28), pygame.Rect(411, 119 + i \* 33, 80, 28), pygame.Rect(496, 119 + i \* 33, 80, 28), pygame.Rect(581, 119  + i \* 33, 80, 28)])

        self.txt = [self.font.render("Name",True, BLACK), self.font.render("Time",True, BLACK), self.font.render("Deaths",True, BLACK), self.font.render("level",True, BLACK)]

        self.txt.extend([self.font.render(str(item), True, BLACK) for sub in res1 for item in sub])

        self.txt.extend([self.font.render(str(item), True, BLACK) for sub in res2 for item in sub])

    def drawScreen(self, screen: pygame.Surface):

        for sprite in self.instruction\_sprite\_group:

            sprite.draw(screen)

        for txt, rect in zip(self.txt, self.whitebox):

            pygame.draw.rect(screen, WHITE, rect, 0)

            screen.blit(txt, (rect.left + 5, rect.top + 4))

It consists of boxes and text, where the information is loaded from the SQLite database while only getting the top 3 records for each level in terms of time.

### Iteration 6 Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected Output | Acutal Output | Solution |
| Press space | One bullet is produced | As expected |  |
| Constantly spamming space bar | Bullets are produced in a interval | As expected |  |
| Bullet hit enemy | Enemy reduce in health | As expected |  |
| Collide with the finish flag (win the game) | Victory screen shown with correct time and death number | As expected |  |
| Click on ranking screen button | Ranking screen shown | As expected |  |
|  | Rankings in ascending order of time | As expected |  |

#### Success Criteria

* The game show a victory screen when the player wins, showing the time used and death number
* The game will have a ranking screen showing the top entries of each level
* Ranking board button
* A SQLite table to store all the successful entries
* The character can shoot the enemies, and enemies will reduce in health when the bullet hits them
* The character can shoot horizontally

# Testing results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | Function | Scenario | Expected Output | Actual result | Screenshot reference |
| 1 | Player movement | “A” pressed | Character move left | As expected | 1 -> 4 |
| 2 | “D” pressed | Character move right | As expected | 1 -> 5 |
| 3 | “W” pressed | Character jumps | As expected | 1 -> 2 |
| 4 | “W” pressed x 2 | Character jumps twice | As expected | 1 -> 3 |
| 5 | “W” pressed x 3 | Character jump twice and not jump and third press | As expected | 1 -> 3 |
| 6 | Checkpoint | Character move over the checkpoint and dies | The player respawn at checkpoint | As expected | 6 -> 7 -> 8 |
| 7 | Restart | Press restart through in-game menu | Restart and the beginning of the level | As expected |  |
| 8 | Ranking board |  | Screen displaying top 3 results of each level | As expected |  |
| 9 | Death | Player HP reduced to zero | Character dies, print death screen and respawn at checkpoint | As expected |  |
| 10 | Collision with trap | Character collide with traps | Character dies and respawn at checkpoint | As expected |  |
| 11 | Collision with enemies | Character collide with enemies | Character dies and respawn at checkpoint | As expected |  |
| 12 | Falling off the map | Character fall off the map | Character dies and respawn | As expected |  |
| 13 | Win | Character collide with flag pole | Win screen and return to the main menu | As expected |  |
| 14 | Camera Movement | Character tries to move off of the camera | The camera follows the character | As expected |  |
| 15 | Shoot Bullet | Pressed space bar | Bullet shoot off of the character | As expected |  |
| 16 | Bullet collision with enemies | Bullet hit enemies | Enemies health reduce | As expected |  |
| 17 | Enemies death | Enemies HP reduce to zero | Enemies death | As expected |  |
| 18 | Bullet collision with tiles | Bullet hit tiles | Bullet dies | As expected |  |
| 19 | Traps activation | Character move to activation zone | Trap get activated | As expected |  |
| 20 | Menu class |  | A menu is shown with 3 level buttons and 1 setting Button | As expected |  |
| 21 | Mouse response / level button class | Left-click on each level button | Print level button mouse collided | As expected |  |
| 22 | Mouse response / menu button class | Left-click on the menu button | Blank menu appears at the middle of the screen | As expected |  |
| 23 | Level button class / initialise game | Click on each level buttons | Screen change to the corresponding level | As expected |  |
| 24 | Parse map |  | Map loaded | As expected |  |
| 25 | Game class/ player class |  | character loaded | As expected |  |
| 26 | Menu Button | Click on the in-game menu button | In-game menu popup | As expected |  |
| 27 | Quit game button | Click on exit game button | Exit game | As expected |  |
| 28 | Player class / movement |  | Player doesn’t fall off the floor | As expected |  |
| 29 | Player class / movement | Move right and collide | Player stop when colliding | As expected |  |
| 30 | Player class / movement | Move left and collide | Player stop when colliding | As expected |  |
| 31 | Player class / movement | Move off an edge | Fall off | As expected |  |
| 32 | Player class / jump | Press “W” twice | Double jump | As expected |  |
| 33 | Player class / jump | Press “W” twice after falling off an edge | Double jump | As expected |  |
| 34 | Player class / movement | Fall off the floor | Player dies and present death screen | As expected |  |
| 35 | Menu class | Press menu button | Menu pop up | As expected |  |
| 36 | Instruction screen class | Press instruction button inside the in-game menu | Instructions menu pop up | As expected |  |
| 37 | Restart button class | Press restart button | Game restart | As expected |  |
| 38 | Exit game button | Press exit game button | Exit to the main menu | As expected |  |
| 39 | Trap Group / parse traps | Load trap csv | All trap presented on the screen | As expected |  |
| 40 | Player / enemy interaction | Player touches Spike | Player dies | As expected |  |
| 41 | Spike class / activation / logic | Player collide with spike up activation zone | Spike up moves as designed | As expected |  |
| 42 | Player / enemy interaction | Player collide with spike up | Player dies | As expected |  |
| 43 | Appearblock class / activation | Player collide with appear block | Block appears | As expected |  |
| 44 | Disappear block class / activation | Player Collide with disappear block | Block disappear and no collision | As expected |  |
| 45 | Grow spike class / activation / logic | Player collide with grow spike activation zone | Spike change in size | As expected |  |
| 46 | Player / enemy interaction | Player collide with activated grow spike | Player dies | As expected |  |
| 47 | Player / enemy interaction | Player collide with horizontal spike | Player dies | As expected |  |
| 48 | Moving horizontal spike class / activation / logic | Player collide with moving horizontal spike activation zone | Spike moves | As expected |  |
| 49 | Finish flag class | Player collide with finish flag | Game ends | As expected |  |
| 50 | Checkpoint class | Player collide with checkpoint and dies | Player respawn in the checkpoint after a few seconds | As expected |  |
| 51 | Goomba class |  | Goomba moves within the designated area on its own | As expected |  |
| 52 | Player / enemy interaction | Player collide with Goomba | Player dies | As expected |  |
| 53 | Player / shoot bullet | Press space | One bullet is produced | As expected |  |
| 54 | Player / shoot bullet / shoot timer | Constantly spamming space bar | Bullets are produced in a interval | As expected |  |
| 55 | Player / bullet enemy interaction | Bullet hit enemy | Enemy reduce in health | As expected |  |
| 56 | Victory screen class | Collide with the finish flag (win the game) | Victory screen shown with correct time and death number | As expected |  |
| 57 | Ranking button class | Click on ranking screen button | Ranking screen shown | As expected |  |
| 58 | Ranking screen class |  | Rankings in ascending order of time | As expected |  |

## Screenshot

A picture containing shape

Description automatically generatedA picture containing logo

Description automatically generated

Figure 2

A picture containing shape

Description automatically generated

Figure 4

Figure 1

A picture containing shape

Description automatically generatedA picture containing logo

Description automatically generated

Figure 6

Figure 3

A picture containing graphical user interface

Description automatically generated

Figure 5

A picture containing shape

Description automatically generatedShape

Description automatically generated with low confidence

Figure 8

Figure 7

## Result of testing

All designated features and functions have passed the test. However, in one of the playthrough I have done, the player somehow teleport through a tile diagonally. Unfortunately, the bug rarely occurs and I was not able to recreate the problem and thus cannot find the solution to this problem.

Furthermore, I have faced some issues with the Pygame module which does not support Python 11. Instead, I have to install the pre-released version of Pygame that is compatible with Python 11.

When I was trying to run the game file in another computer with different operating system, it cannot find the relative path using string. Instead, I change all the pathway strings to os.path.join()

Example:

Old version

self.parseEntities(f"files/entity{str(level)}.csv")

New version

self.parseEntities(os.path.join("files","entity"+str(level)+".csv"))

During game play, I have noticed that sometime, the enemies will have different speed even though their movement speed is set to constant. After testing with different value, I found that for the position of the objects, it will automatically round up the position if it has a decimal place. As a result, the enemy will have a higher speed when moving in a positive direction but slower in a negative direction. At the beginning, I set the speed to a number relative to the size of the block so that I don’t need to change all the value if I decide to change the block size, which result in floating point numbers and different in actual speed. After all things are set and done, I change all values to constant integer so that no floating point numbers will be produced.

## User Testing

I will send the game file to my user. They will test the game according to the question stated below.

|  |  |  |
| --- | --- | --- |
| Question | Answer | Remark |
| Can you run the game? | Yes(10)  No(0) |  |
| Can you start each level? | Yes(10)  No(0) |  |
| Can you move horizontally? | Yes(10)  No(0) |  |
| Can you double jump? | Yes(10)  No(0) |  |
| Can you win the game? | Yes(10)  No(0) |  |
| Is the record shown in the ranking board | Yes(10)  No(0) |  |
| Do all the buttons work? | Yes(10)  No(0) |  |
| Does the size of the character suitable for the game? | Yes(10)  No(0) |  |
| Are the maps too difficult or too easy? | Yes(1)  No(9) |  |
| Can you see enemies health bar? | Yes(10)  No(0) |  |
| Can you activate the traps? | Yes(10)  No(0) |  |
| Does the colour theme suits your expectation? | Yes(7)  No(3) |  |
| Is the game fun? | Yes(9)  No(1) |  |
| Do you experience lagging while playing the game? | Yes(0)  No(10) |  |

## Final version of the code

#### Main.py

import sys

import pygame

from constants import \*

from helper import \*

from mario import Game

import sqlite3

con = sqlite3.connect("data.db", check\_same\_thread=False)

db = con.cursor()

class Main:

    def \_\_init\_\_(self):

        self.size = SIZE

        self.screen = pygame.display.set\_mode(self.size)

        pygame.display.set\_caption("Mario")

        self.clock = pygame.time.Clock()

        self.all\_sprites\_list = []

        self.currentScreen = MENU

        self.player\_name = ""

        # Objects

        self.menu = Menu(self.screen)

        self.setting = MenuSettingScreen()

        self.instructionScreen = InstructionScreen(SCREENTOGAMEMENU)

        self.rankingScreen = RankingScreen(SCREENTOGAMEMENU)

        self.all\_sprites\_list.extend([self.menu, self.setting])

        # Sprite Groups

        self.status = []

    def keyResponse(self,event: pygame.event):

        if event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_1:

                self.status.append(create\_level\_status\_code(1))

            elif event.key == pygame.K\_2:

                self.status.append(create\_level\_status\_code(2))

    # Change status (Task pending to do)

    def mouseResponse(self, position: tuple, click):

        if self.currentScreen == MENU:

            self.status = self.menu.mouseInteraction(position, self.status, click)

        elif self.currentScreen == SETTINGSCREEN:

            self.status = self.setting.mouseInteraction(position, self.status, click)

        elif self.currentScreen == INSTRUCTIONSCREEN:

            self.status = self.instructionScreen.mouseInteraction(position, self.status, click)

        elif self.currentScreen == RANKINGSCREEN:

            self.status = self.rankingScreen.mouseInteraction(position, self.status, click)

    def setCurrentScreen(self, newScreen: str):

        self.currentScreen = newScreen

    def initialiseGame(self, level: int):

        game = Game(level)

        restart, respawn\_checkpoint, check\_point, death\_count, time\_count, record = game.play()

        if record:

            db.execute(f"INSERT INTO Entries (name, time, deaths, level) VALUES ('{self.player\_name}', {time\_count}, {death\_count}, {level});")

            con.commit()

            death\_count = 0

            time\_count = 0

        elif not respawn\_checkpoint:

            death\_count = 0

            time\_count = 0

        while restart:

            game = Game(level, respawn = respawn\_checkpoint, check\_point = check\_point, death\_count = death\_count, time\_count = time\_count)

            restart, respawn\_checkpoint, check\_point, death\_count, time\_count, record = game.play()

            if record:

                db.execute(f"INSERT INTO Entries (name, time, deaths, level) VALUES ('{self.player\_name}', {time\_count}, {death\_count}, {level});")

                con.commit()

                death\_count = 0

                time\_count = 0

            elif not respawn\_checkpoint:

                death\_count = 0

                time\_count = 0

    def readStatus(self):

        for stat in self.status:

            # Change to menu screen

            if stat == SCREENTOSETTING:

                self.setCurrentScreen(SETTINGSCREEN)

            # Change to setting screen

            elif stat == SCREENTOGAMEMENU:

                self.setCurrentScreen(MENU)

            # Initialise game

            elif check\_status\_init\_level(stat):

                level = extract\_level\_from\_status\_code(stat)

                self.initialiseGame(level)

            # Exit game

            elif stat == EXITGAME:

                sys.exit()

            # Change to instruction screen

            elif stat == SCREENTOINSTRUCTION:

                self.setCurrentScreen(INSTRUCTIONSCREEN)

            elif stat == SCREENTORANKING:

                self.rankingScreen.refresh()

                self.setCurrentScreen(RANKINGSCREEN)

        self.status = []

    def logic(self):

        pass

    def drawScreen(self):

        self.screen.fill(BROWN)

        if self.currentScreen == MENU:

            self.menu.drawScreen(self.screen)

        elif self.currentScreen == SETTINGSCREEN:

            self.setting.drawScreen(self.screen)

        elif self.currentScreen == INSTRUCTIONSCREEN:

            self.instructionScreen.drawScreen(self.screen)

        elif self.currentScreen == RANKINGSCREEN:

            self.rankingScreen.drawScreen(self.screen)

        pygame.display.flip()

    def gameplay(self):

        self.inputName()

        done = False

        while not done:

            # -- User input and controls

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    sys.exit()

                self.keyResponse(event)

            click,\_,\_ = pygame.mouse.get\_pressed()

            mouse = pygame.mouse.get\_pos()

            self.mouseResponse(mouse,click)

            self.readStatus()

            #--Game logic goes after this comment

            self.logic()

            # -- Screen background is BLACK

            self.drawScreen()

            self.clock.tick(60)

        #Endwhile

        self.finishScreen()

    def inputName(self):

        self.font = pygame.font.Font("freesansbold.ttf",100)

        self.nameStr = self.font.render("Name: ", True, WHITE)

        self.nameStr\_pos = (182, 80)

        self.name\_pos = (182, 80 + self.nameStr.get\_size()[1])

        temp\_name = ""

        done = False

        while not done:

            for event in pygame.event.get():

                    if event.type == pygame.QUIT:

                        sys.exit()

                    elif event.type == pygame.KEYDOWN:

                        if event.key == pygame.K\_RETURN:

                            self.player\_name = temp\_name

                            done = True

                        elif event.key == pygame.K\_BACKSPACE:

                            temp\_name = temp\_name[:-1]

                        else:

                            temp\_name += event.unicode

            self.screen.fill(BROWN)

            self.screen.blit(self.nameStr, self.nameStr\_pos)

            self.screen.blit(self.font.render(temp\_name, True, WHITE), self.name\_pos)

            pygame.display.flip()

#Menu Screen

class Menu():

    def \_\_init\_\_(self, screen: pygame.Surface):

        self.size = SIZE

        self.screen = screen

        self.menuSpriteGroup = pygame.sprite.Group()

        self.levelButtonGroup = []

        self.settingButton = SettingButton(\*SETTINGS\_BUTTON\_SIZE, (SIZE[0] - 132, 20))

        self.menuSpriteGroup.add(self.settingButton)

        startY = 120 + LEVEL\_IMAGE\_SIZE[1] + LEVEL\_IMAGE\_BUTTON\_PADDING

        startX = (SIZE[0] - NUM\_OF\_LEVELS \* LEVEL\_IMAGE\_SIZE[0] + (NUM\_OF\_LEVELS - 1) \* LEVEL\_IMAGE\_BUTTON\_PADDING)/2

        self.img1 = pygame.transform.scale(pygame.image.load(os.path.join("images", "Level1.png")), (240, 168))

        self.img2 = pygame.transform.scale(pygame.image.load(os.path.join("images", "Level2.png")), (240, 168))

        self.img1\_pos = ((SIZE[0] - NUM\_OF\_LEVELS \* LEVEL\_IMAGE\_SIZE[0] + (NUM\_OF\_LEVELS - 1) \* LEVEL\_IMAGE\_BUTTON\_PADDING)/2 - 46,120)

        self.img2\_pos = (self.img1\_pos[0] + LEVEL\_2\_LEVEL\_PADDING + LEVEL\_IMAGE\_SIZE[0],120)

        for lvlNum in range(NUM\_OF\_LEVELS):

            levelBut = LevelButton(lvlNum+1, f"Level {lvlNum + 1}",\*LEVEL\_BUTTON\_SIZE,(startX,startY))

            self.levelButtonGroup.append(levelBut)

            startX += LEVEL\_2\_LEVEL\_PADDING + LEVEL\_IMAGE\_SIZE[0]

    # Return new status

    def mouseInteraction(self, position: tuple, status: list, click: bool):

        for sprite in self.menuSpriteGroup:

            status = sprite.mouseInteraction(position, status, click)

        for but in self.levelButtonGroup:

            status = but.mouseInteraction(position, status, click)

        return status

    def drawScreen(self,screen: pygame.Surface):

        self.settingButton.draw(screen)

        for sprite in self.menuSpriteGroup:

            sprite.draw(screen)

        for but in self.levelButtonGroup:

            but.draw(screen)

        screen.blit(self.img1, self.img1\_pos)

        screen.blit(self.img2, self.img2\_pos)

class LevelButton(WordButton):

    def \_\_init\_\_(self,level: int, name: str, width: int, height: int, position: tuple[int]):

        super().\_\_init\_\_(width, height, position, WHITE, BLACK, "Level "+str(level), create\_level\_status\_code(level))

        self.name = name

        self.level = level

# Setting Screen

class MenuSettingScreen:

    def \_\_init\_\_(self):

        self.background = Background(648,336,(76, 64), SETTINGSCREENCOLOR)

        self.closeButton = CloseButton(24,24, (712, 52), SCREENTOGAMEMENU)

        self.exitButton = QuitGameButton((331, 306))

        self.controlButton = InstructionButton((331,118))

        self.rankingButton = RankScreenButton((331,218))

        self.gameMenu\_sprite\_group = [self.background, self.closeButton, self.exitButton,self.controlButton, self.rankingButton]

    def drawScreen(self, screen: pygame.Surface):

        for sprite in self.gameMenu\_sprite\_group:

            sprite.draw(screen)

    def mouseInteraction(self,position: tuple, status: list[str], click):

        for sprite in self.gameMenu\_sprite\_group:

            status = sprite.mouseInteraction(position, status, click)

        return status

    def keyResponse(self,event: pygame.event, status: list[str]):

        for sprite in self.gameMenu\_sprite\_group:

            status = sprite.keyResponse(event, status)

        return status

class RankingScreen:

    def \_\_init\_\_(self, closeStatusCode: str):

        self.font = pygame.font.Font("freesansbold.ttf", FONTSIZE)

        self.background = Background(648,336,(76, 64), SETTINGSCREENCOLOR)

        self.closeButton = CloseButton(24,24, (712, 52), closeStatusCode)

        self.instruction\_sprite\_group = [self.background, self.closeButton]

        self.background = pygame.Rect(134,114,532,237)

        self.refresh()

    def mouseInteraction(self, position: tuple, status: list, click):

        return self.closeButton.mouseInteraction(position, status, click)

    def keyResponse(self,event: pygame.event, status: list):

        return status

    def refresh(self):

        res1 = db.execute("SELECT Name, Time, Deaths, level FROM Entries WHERE level = 1 ORDER BY Time, Deaths ASC").fetchall()[0:3]

        res2 = db.execute("SELECT Name, Time, Deaths, level FROM Entries WHERE level = 2 ORDER BY Time, Deaths ASC").fetchall()[0:3]

        self.whitebox = []

        for i in range(7):

            self.whitebox.extend([pygame.Rect(139, 119 + i \* 33, 267, 28), pygame.Rect(411, 119 + i \* 33, 80, 28), pygame.Rect(496, 119 + i \* 33, 80, 28), pygame.Rect(581, 119  + i \* 33, 80, 28)])

        self.txt = [self.font.render("Name",True, BLACK), self.font.render("Time",True, BLACK), self.font.render("Deaths",True, BLACK), self.font.render("level",True, BLACK)]

        self.txt.extend([self.font.render(str(item), True, BLACK) for sub in res1 for item in sub])

        self.txt.extend([self.font.render(str(item), True, BLACK) for sub in res2 for item in sub])

    def drawScreen(self, screen: pygame.Surface):

        for sprite in self.instruction\_sprite\_group:

            sprite.draw(screen)

        for txt, rect in zip(self.txt, self.whitebox):

            pygame.draw.rect(screen, WHITE, rect, 0)

            screen.blit(txt, (rect.left + 5, rect.top + 4))

if \_\_name\_\_ == "\_\_main\_\_":

    pygame.init()

    main = Main()

    main.gameplay()

    pygame.quit()

#### Mario.py

import csv

import os

import sys

import pygame

from camera import \*

from constants import \*

from helper import \*

from map1 import \*

from map2 import \*

class Game:

    def \_\_init\_\_(self, level = 1, respawn = False, check\_point = None, death\_count = 0, time\_count = 0):

        # Pygame setups

        self.size = SIZE # (810,540)

        self.screen = pygame.display.set\_mode(self.size)

        self.clock = pygame.time.Clock()

        self.done = False

        # Load Map

        # self.map = Map(os.path.join("files","map" + str(level) +".csv"))

        self.map = Map(level)

        # Level of this game

        self.level = level

        # Menu Buttons

        self.menu\_button = MenuButton()

        # Game Menu Screen

        self.menuScreen = GameMenuScreen()

        # Status Codes

        self.status = []

        # Instruction Screen (from menu screen)

        self.instruction\_screen = InstructionScreen(RETURNTOGAME)

        # Game Board

        self.game\_board = GameBoard(death\_count = death\_count, time\_count = time\_count)

        self.time\_count = time\_count

        # Set Current Screen Code

        self.currentScreen = GAME

        # Let the main know if the game is restarting or quiting

        self.restart = False

        # Text Shown when death

        self.death\_text = DeathText()

        self.win\_text = WinText()

        # Time count until automatic restart from checkpoint

        self.death\_count = 0

        self.death\_count\_start = False

        # Time count until automatic exit

        self.win\_count = 0

        self.win\_count\_start = False

        # Sprite Group that contains everything

        self.gameSpriteGroup = []

        self.record = True

        # Check if it is restarting from checkpoint

        self.respawn\_checkpoint = True

        if respawn:

            self.check\_point = check\_point

        else:

            self.check\_point = self.map.player\_start\_pos

        # Create Player object

        self.player = Player(HORIZONAL\_MAX\_SPEED, self.map.vertices, self.check\_point)

        # Create Camera (containing the position of the camera)

        self.camera = Camera(self.player, self.map.vertices)

        # Enemy sprites group

        self.enemy\_sprite\_group = pygame.sprite.Group()

        self.gameSpriteGroup.extend([self.menu\_button])

        # Parse in the enemy from flat file

        self.parseEntities(os.path.join("files","entity"+str(level)+".csv"))

        # Create all traps

        self.trap\_group = TrapGroup(os.path.join("files","trap"+str(level)+".csv"))

        self.gameSpriteGroup.extend([self.trap\_group, self.game\_board])

    def parseEntities(self, filename: str):

        """

        Parse different enemies to add into game sprite group

        Args:

            filename (str): name of the file

        """

        with open(filename, 'r') as f:

            reader =  csv.reader(f)

            next(reader)

            for row in reader:

                self.create\_enemies(row)

        self.gameSpriteGroup.extend(self.enemy\_sprite\_group)

    def create\_enemies(self, row: list):

        """

        Create Enemy object from each line and add into enemy sprite group

        Args:

            row (list): row in the csv file

        """

        # Change relative position to pygame coordinates

        position = relativeCoor2DeCoor((int(row[1]), int(row[2])))

        # Interprete the boundaries

        x\_boundary = tuple(apply(row[3].split("|"), xReToDe))

        y\_boundary = tuple(apply(row[4].split("|"), yReToDe))

        # Create different enemies

        if row[0] == "goomba":

            temp = Goomba(position, x\_boundary, y\_boundary)

            self.enemy\_sprite\_group.add(temp)

    def gameResponse(self, event: pygame.event):

        """

        The game response to different events

        Args:

            event (pygame.event): pygame event

        """

        if event.type == pygame.KEYDOWN:

            # When it is inside the death screen + enter is pressed, the game restart instantly

            if event.key in [pygame.K\_RETURN, pygame.K\_SPACE] and self.death\_count\_start:

                self.done = True

                self.restart = True

                self.respawn\_checkpoint = True

                self.record = False

            elif event.key == pygame.K\_ESCAPE:

                if self.currentScreen == GAME:

                    self.currentScreen = GAMEMENU

                else:

                    self.currentScreen = GAME

            elif event.key == pygame.K\_r:

                self.done = True

                self.restart = True

                self.respawn\_checkpoint = False

                self.record = False

    def keyResponse(self, event: pygame.event):

        """

        Response to different keys being pressed

        Args:

            event (pygame.event): pygame event

        """

        # Different response with respect to the screen mode

        if self.currentScreen == GAMEMENU:

            self.status = self.menuScreen.keyResponse(event,self.status)

        elif self.currentScreen == INSTRUCTIONSCREEN:

            self.status = self.instruction\_screen.keyResponse(event, self.status)

        elif self.currentScreen == GAME:

            self.status = self.player.keyResponse(event, self.status)

        self.gameResponse(event)

    def mouseResponse(self, position: tuple, click):

        """

        Response to mouse cursor clicks

        Args:

            position (tuple): position of the mouse pointer

        """

        self.status = self.menu\_button.mouseInteraction(position, self.status, click)

        # Different response according to the screen mode

        if self.currentScreen == GAMEMENU:

            self.status = self.menuScreen.mouseInteraction(position, self.status, click)

        elif self.currentScreen == INSTRUCTIONSCREEN:

            self.status = self.instruction\_screen.mouseInteraction(position, self.status, click)

    def readStatus(self):

        """

        Read the status code of the game

        """

        for stat in self.status:

            # Turn screen mode to game menu

            if stat == SCREENTOGAMEMENU:

                self.currentScreen = GAMEMENU

            # Exit the current game

            elif stat == CLOSEGAME:

                self.done = True

                self.record = False

            # Turn screen mode to game mode

            elif stat == RETURNTOGAME:

                self.currentScreen = GAME

            # Turn screen mode to instruction screen

            elif stat == SCREENTOINSTRUCTION:

                self.currentScreen = INSTRUCTIONSCREEN

            # Restart the current game

            elif stat == RESTARTGAME:

                self.done = True

                self.restart = True

                self.respawn\_checkpoint = False

                self.record = False

            # Turn the screen mode to death screen

            elif stat == PLAYERDEATH:

                self.currentScreen = DEATHSCREEN

                self.time\_count = (pygame.time.get\_ticks() - self.game\_board.start\_time)//1000 + self.game\_board.time\_count

                self.death\_count\_start = True

                self.game\_board.death\_count += 1

            elif stat == PLAYERWIN:

                self.currentScreen = WINSCREEN

                self.time\_count = (pygame.time.get\_ticks() - self.game\_board.start\_time)//1000 + self.game\_board.time\_count

                self.win\_text.set\_time\_death(self.time\_count, self.game\_board.death\_count)

                self.win\_count\_start = True

        # Reset the screen status for next iteration

        self.status = []

    def logic(self):

        """

        The main logic of the game

        """

        # Read status code from last iteration

        self.readStatus()

        # Player death screen count down

        self.playerdeath()

        self.playerwin()

        # Only run if the player is playing the game (All game object logic will not work if the menu is opened)

        if self.currentScreen == GAME:

            # Player and bullet interactions with dead zone, tiles and enemies

            self.player.update(self.map.tileGroup, self.map.dead\_zone, self.status,self.enemy\_sprite\_group, self.trap\_group, self.map.disappear\_tiles, self.map.vertices[0:2])

            self.status.append(self.player.checkdeath())

            # Change the camera position according to the player's position

            self.camera.scroll()

            # Check if the player touches a checkpoint block (and set new checkpoint if yes)

            for tile in self.map.checkpoint\_group:

                res = tile.player\_interaction(player\_rect = self.player.rect)

                if res != None:

                    self.check\_point = res

            if self.player.rect.colliderect(self.map.finish\_point.rect):

                self.status.append(PLAYERWIN)

            # Player and tiles interaction (Nothing for now)

            for tile in self.map.tileGroup:

                tile.player\_interaction(player\_rect = self.player.rect)

            # Traps logic

            self.trap\_group.logic()

            for tiles in self.map.disappear\_tiles:

                tiles.logic()

            # Update enemies

            for enemy in self.enemy\_sprite\_group:

                enemy.update()

                if enemy.hp <= 0:

                    enemy.kill()

                    self.gameSpriteGroup.remove(enemy)

            # Update game board (Nothing for now)

            self.game\_board.logic()

    def drawScreen(self):

        """

        Draw Screen

        """

        # fill background as sky blue

        self.screen.fill(SKYBLUE)

        # Draw everything

        for sprite in self.gameSpriteGroup:

            sprite.draw(self.screen, self.camera.position)

        self.map.draw(self.screen, self.camera.position)

        self.player.draw(self.screen, self.camera.position)

        # Draw menu screen on top

        if self.currentScreen == GAMEMENU:

            self.menuScreen.drawScreen(self.screen)

        elif self.currentScreen == INSTRUCTIONSCREEN:

            self.instruction\_screen.drawScreen(self.screen)

        elif self.currentScreen == DEATHSCREEN:

            self.death\_text.draw(self.screen)

        elif self.currentScreen == WINSCREEN:

            self.win\_text.draw(self.screen)

        pygame.display.flip()

    def playerdeath(self):

        """

        Count down for the death screen to disappear

        """

        if self.death\_count\_start:

            self.death\_count +=1

        if self.death\_count >= 200:

            self.done = True

            self.restart = True

            self.respawn\_checkpoint  = True

    def playerwin(self):

        """

        Count down for the win screen to disappear and exit the game

        """

        if self.win\_count\_start:

            self.win\_count += 1

        if self.win\_count >= 200:

            self.done = True

            self.restart = False

            self.respawn\_checkpoint = False

    def play(self) -> tuple[bool, bool, tuple, int] :

        """

        Main Game loop

        Returns:

            Restart (bool): If the game should restart or not

            Respawn Checkpoint (bool): If the player should respawn in the check point

            Checkpoint (tuple): tuple for the coordinate of the checkpoint

            Death count: the death count of the player if he / she respawn in checkpoint instead of restarting

        """

        self.done = False

        while not self.done:

            logging.debug(f"{self.player.rect.x}")

            # Loop throught events

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    self.done = True

                    sys.exit()

                self.keyResponse(event)

            # Cursor interactions

            click,\_,\_ = pygame.mouse.get\_pressed()

            mouse = pygame.mouse.get\_pos()

            self.mouseResponse(mouse, click)

            # Game logics

            self.logic()

            # Draw Screen

            self.drawScreen()

            # 60 ticks per second

            self.clock.tick(60)

        return self.restart, self.respawn\_checkpoint, self.check\_point, self.game\_board.death\_count,self.time\_count, self.record

# Player object

class Player(pygame.sprite.Sprite):

    """

    Player object

    """

    def \_\_init\_\_(self, horizonal\_max\_speed: int, vertices: list, start\_pos: list):

        super().\_\_init\_\_()

        # Import image for the player

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images", "redRect.png")), PLAYER\_SIZE)

        # Create rectangle object for the player

        self.rect = self.image.get\_rect()

        # Intialise it in the setted starting position

        self.rect.x = start\_pos[0]

        self.rect.y = start\_pos[1]

        # Set initial speed as 0

        self.xSpeed = 0

        self.ySpeed = 0

        # Set the boarder of the player

        self.left\_border = vertices[0]

        self.right\_border = vertices[1] - PLAYER\_SIZE[0] # right most of the map

        # Set the maximum horizonal speed the player can travel

        self.horizonal\_max\_speed = horizonal\_max\_speed

        # Set the player status to be on the ground

        self.on\_ground = True

        # Set the health of the player

        self.health = 100

        # Set the direction the player is facing (for animation and shooting of the bullets)

        self.face\_direction = 1

        # Bullet group and timer for shooting the next bullet

        self.bullet\_group = pygame.sprite.Group()

        self.bullet\_timer = 0

        self.bullet\_available = True

    def draw(self, screen, cam\_position):

        """

        Draw the player

        Args:

            screen (\_type\_): screen to draw on

            cam\_position (\_type\_): the position of the camera

        """

        screen.blit(self.image, (self.rect.x - cam\_position.x, self.rect.y))

        # Draw bullets

        for bullet in self.bullet\_group:

            bullet.draw(screen, cam\_position)

    def set\_speed\_x(self, speed: int):

        """

        Set the horizontal speed of the player

        """

        self.xSpeed = speed

    def shoot\_bullet(self):

        """

        Shoot bullet

        """

        # Check if the player is allow to shoot bullets

        if self.bullet\_available:

            bullet = Bullet(self.face\_direction, self.rect.center)

            self.bullet\_group.add(bullet)

            self.bullet\_available = False

    # Timer for time between each bullet

    def shoot\_timer(self):

        if not self.bullet\_available:

            self.bullet\_timer += 1

        if self.bullet\_timer >= BULLET\_TIMER:

            self.bullet\_available = True

            self.bullet\_timer = 0

    # Interactions between bullets and enemies

    def bullet\_enemy\_interaction(self, enemies: list):

        for enemy in enemies:

            hits = pygame.sprite.spritecollide(enemy, self.bullet\_group, False)

            enemy.change\_hp(-len(hits) \* BULLET\_DAMAGE)

            for bu in hits:

                bu.kill()

    def jump(self, hack: bool = False):

        """

        Control the logic of jumping

        """

        # If the player is on the ground, the player jump

        if self.on\_ground:

            self.jump\_num = 1

            self.on\_ground = False

            self.ySpeed = -JUMP\_SPEED

        # If the player is on air and there is jumps available, the player jump and reduce the number of jumps available

        elif self.jump\_num > 0 or hack:

            self.jump\_num -= 1

            self.on\_ground = False

            self.ySpeed = -JUMP\_SPEED

    def movementX(self):

        """

        Horizonal movement logic

        """

        self.rect.x += self.xSpeed

        # boarder collision

        self.rect.x = min(self.rect.x, self.right\_border)

        self.rect.x = max(self.rect.x, self.left\_border)

    def movementY(self):

        """

        Vertical movement logic

        """

        self.rect.y += self.ySpeed

        self.ySpeed += GRAVITY

    def collisionX(self, tiles: list):

        """

        Horizonal interaction with tiles it collided with

        Args:

            tiles (list): The ground tiles

        """

        # For moving right

        if self.xSpeed > 0:

            # If there is collision, change x coordinate to the lef to the block it collide with

            for tile in tiles:

                self.rect.x = min(tile.rect.x - PLAYER\_SIZE[0], self.rect.x)

        # For moving left

        elif self.xSpeed < 0:

            # IF there are collisions, change x coordinate to the right of the block it collide with

            for tile in tiles:

                self.rect.x = max(tile.rect.right, self.rect.x)

    def collisionY(self, tiles: list):

        """

        Veritical interaction with tiles it collided with

        Args:

            tiles (list): Ground tiles

        """

        # Moving down

        if self.ySpeed > 0:

            self.on\_ground = True

            # Set the y coordinate to the top of the collide blocks

            for tile in tiles:

                self.rect.y = min(tile.rect.top - PLAYER\_SIZE[1], self.rect.y)

        # Moving up

        elif self.ySpeed < 0:

            # Set the y coordinate to the top of the collide blocks

            for tile in tiles:

                self.rect.y = max(tile.rect.bottom, self.rect.y)

        # Taking away any vertical velocity if collision

        self.ySpeed = 0

    def enemy\_interaction(self,enemy\_sprite\_group: list, trap\_group: TrapGroup) -> str:

        """

        Interaction between enemy, traps and player

        Args:

            enemy\_sprite\_group (list): Group of enemy sprite

            trap\_group (TrapGroup): Traps

        Returns:

            str: Status Code

        """

        # Player dies if it touches with any enemy

        for enemy in enemy\_sprite\_group:

            if type(enemy) == Goomba:

                if pygame.sprite.collide\_rect(self, enemy):

                    self.health -= 200

                    return

        # Player dies if it touches traps

        for trap in trap\_group.all\_trap\_group:

            if type(trap) in [Spike, SpikeUp,GrowSpike, HorizontalSpike,MoveableHoriSpike]:

                for rect in trap.rect\_group:

                    if self.rect.colliderect(rect):

                        self.health -= 200

                        return

        # Return None if the player hasn't died

        return None

    # Logic and updates

    def update(self, tiles: list, dead\_zone: list, status: str, enemy\_sprite\_group: pygame.sprite.Group, trap\_group: TrapGroup, disappear\_tiles: list, x\_boundary: list) -> bool:

        """

        Update the Player including the interaction with environment

        Args:

            tiles (list): List of tiles

            dead\_zone (list): list of the dead zones

            status (str): status code

            enemy\_sprite\_group (pygame.sprite.Group): Enemy sprite group

            trap\_group (TrapGroup): Traps

        Returns:

            (bool): If the player is dead

        """

        death = False

        # Horizonal movements and collisions

        self.movementX()

        collided\_tiles = pygame.sprite.spritecollide(self, tiles, False)

        if len(collided\_tiles) != 0:

            self.collisionX(collided\_tiles)

        # Vertical movements and collisions

        self.movementY()

        collided\_tiles = pygame.sprite.spritecollide(self, tiles, False)

        if len(collided\_tiles) != 0:

            self.collisionY(collided\_tiles)

        # Shoot logics

        self.shoot\_timer()

        self.bullet\_group.update(tiles, x\_boundary)

        # Traps activaltion

        trap\_group.player\_interaction(self.rect)

        for tile in disappear\_tiles:

            tile.player\_interaction(self.rect)

        # Check if the player touches dead zone

        self.check\_dead\_zone(dead\_zone)

        # Bullet-enemy interaction

        self.bullet\_enemy\_interaction(enemy\_sprite\_group)

        # Player-enemy interaction

        self.enemy\_interaction(enemy\_sprite\_group, trap\_group)

    def check\_dead\_zone(self,dead\_zone: list):

        """

        Check if the player touches the deathzone

        Args:

            dead\_zone (list): dead zones

        """

        for dead in dead\_zone:

            if self.rect.x >= dead[0] and self.rect.x <= dead[0] + dead[2] and self.rect.y >= dead[1] and self.rect.y <= dead[1] + dead[3]:

                self.health -= 1000

    def checkdeath(self) -> str:

        """

        Check if the player has any health left

        Returns:

            (str): if the player is death, it will return a status code indicating the player is dead

        """

        if self.health <= 0:

            return PLAYERDEATH

        return None

    # Key response

    def keyResponse(self,event: pygame.event, status: list) -> list:

        """

        Key response

        Args:

            event (pygame.event): event happening in pygame (key inputs)

            status (list): status code list

        Returns:

            status(list): status codes list

        """

        if event.type == pygame.KEYDOWN:

            # If W is pressed, jump

            if event.key == pygame.K\_w:

                self.jump()

            elif event.key == pygame.K\_s:

                self.jump(True)

            # If space is pressed, shoot bullet

            elif event.key == pygame.K\_SPACE:

                self.shoot\_bullet()

        # get the key being pressed

        key\_p = pygame.key.get\_pressed()

        # If the A is pressed down, set speed as left

        if key\_p[pygame.K\_a]:

            self.set\_speed\_x(-self.horizonal\_max\_speed)

            self.face\_direction = -1

        # If the D is pressed down, set speed as right

        elif key\_p[pygame.K\_d]:

            self.set\_speed\_x(self.horizonal\_max\_speed)

            self.face\_direction = 1

        # If None of A or D is pressed, set speed to 0

        else:

            self.set\_speed\_x(0)

        return status

# 27 x 18 block per map

class Map:

    # Initialise Map object

    def \_\_init\_\_(self, level: int):

        # Ground Tiles

        self.groundSpriteGroup = pygame.sprite.Group()

        # All tiles

        self.tileGroup = pygame.sprite.Group()

        # Checkpoint blocks

        self.checkpoint\_group = []

        self.disappear\_tiles = []

        # Deadzones

        self.dead\_zone = []

        self.finish\_point = NULL

        temp = {}

        exec(f"map\_list = map{level}", globals(), temp)

        exec(f"barrier\_list = barrier{level}", globals(), temp)

        exec(f"vertices = vertices{level}", globals(), temp)

        map\_list = temp["map\_list"]

        barrier\_list = temp["barrier\_list"]

        self.vertices = temp["vertices"]

        self.parseMap(map\_list, barrier\_list)

    def parseMap(self, map: list, barrier\_list: list):

        """

        Parse all tiles in map from the map (tile map)

        Args:

            map (list): a x to y map where x is number of tiles horizontally and y is th number of tiles vertically. (G = ground tile, D = DisappearBlock, A = appear block, a = air tile, F = finish point, C = checkpoint, P = player position, f = Fake spike)

            barrier\_list (list): Death Zone (left, right, top, bottom)

        """

        self.dead\_zone.append((\*relativeCoor2DeCoor(barrier\_list[0:2]), barrier\_list[2]\*BLOCKSIZE[0], barrier\_list[3]\*BLOCKSIZE[1]))

        for r\_num, row in enumerate(map):

            for c\_num, col in enumerate(row):

                if col == "G":

                    ground = Ground(relativeCoor2DeCoor((c\_num, r\_num)))

                    self.groundSpriteGroup.add(ground)

                    self.tileGroup.add(ground)

                elif col == "D":

                    disappear\_tile = DisappearBlock(relativeCoor2DeCoor((c\_num, r\_num)))

                    self.disappear\_tiles.append(disappear\_tile)

                elif col == "A":

                    appear\_tile = Appear\_block(relativeCoor2DeCoor((c\_num, r\_num)))

                    self.tileGroup.add(appear\_tile)

                elif col == "a":

                    airTile = AirTile(relativeCoor2DeCoor((c\_num, r\_num)))

                    self.tileGroup.add(airTile)

                elif col == "F":

                    self.finish\_point = FinishPoint(relativeCoor2DeCoor((c\_num, r\_num-4)))

                elif col == "C":

                    tile = Check\_point(relativeCoor2DeCoor((c\_num, r\_num)))

                    self.checkpoint\_group.append(tile)

                elif col == "P":

                    self.player\_start\_pos = relativeCoor2DeCoor((c\_num, r\_num))

                    self.player\_start\_pos = (self.player\_start\_pos[0], self.player\_start\_pos[1]+BLOCKSIZE[1]-PLAYER\_SIZE[1])

                elif col == "f":

                    fake = FakeSpike(relativeCoor2DeCoor((c\_num, r\_num)))

                    self.tileGroup.add(fake)

    # Initialise Map object

    # def \_\_init\_\_(self, map\_file\_name: str):

    #     # Ground Tiles

    #     self.groundSpriteGroup = pygame.sprite.Group()

    #     # All tiles

    #     self.tileGroup = pygame.sprite.Group()

    #     # Checkpoint blocks

    #     self.checkpoint\_group = []

    #     self.disappear\_tiles = []

    #     # Deadzones

    #     self.dead\_zone = []

    #     self.finish\_point = NULL

    #     # Parse the csv map file

    #     with open(map\_file\_name, 'r') as f:

    #         reader =  csv.reader(f)

    #         # Skip the header

    #         next(reader)

    #         # Set the first row as vertices of the map

    #         self.vertices = apply(next(reader), int) # min x, max x, min y, max y

    #         # Get the starting position of the player

    #         self.player\_start\_pos = apply(next(reader), int)[0:2] #Left x, top y

    #         # Loop through the remaining row

    #         for row in reader:

    #             self.parseMap(row)

    # def parseMap(self,row: list):

    #     """Create objects according to the information of that row

    #     Args:

    #         row (list): a row in the csv file

    #     """

    #     # Get the relative position

    #     relPos = (int(row[1]), int(row[2]))

    #     # Turn relative position into coordinate in the Pygame

    #     dePos = relativeCoor2DeCoor(relPos)

    #     # Number of block extending horizontally and vertically

    #     width = int(row[3])

    #     height = int(row[4])

    #     # Ground block

    #     if row[0] == "ground":

    #         for i in range(width):

    #             for j in range(height):

    #                 groundTile = Ground((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

    #                 self.groundSpriteGroup.add(groundTile)

    #                 self.tileGroup.add(groundTile)

    #     # Air Block

    #     elif row[0] == "airTile":

    #         for i in range(width):

    #             for j in range(height):

    #                 airTile = AirTile((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

    #                 self.tileGroup.add(airTile)

    #     # Deadzone ("barrier")

    #     elif row[0] == "barrier":

    #         self.dead\_zone.append((\*dePos, width\*BLOCKSIZE[0], height\*BLOCKSIZE[1]))

    #     # Hidden block (only appear if the player is close)

    #     elif row[0] == "appear\_block":

    #         for i in range(width):

    #             for j in range(height):

    #                 tile = Appear\_block((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

    #                 self.tileGroup.add(tile)

    #     # Checkpoints

    #     elif row[0] == "check\_point":

    #         tile = Check\_point(dePos)

    #         self.checkpoint\_group.append(tile)

    #     elif row[0] == "finish\_point":

    #         self.finish\_point = FinishPoint(dePos)

        # elif row[0] == "disappear\_block":

        #     for i in range(int(row[3])):

        #         for j in range(int(row[4])):

        #             trap = DisappearBlock((dePos[0]+ BLOCKSIZE[0] \* i, dePos[1]+ BLOCKSIZE[1] \* j))

        #             self.disappear\_tiles.append(trap)

    # Draw objects on screen

    def draw(self,screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        """Draw objects

        Args:

            screen (pygame.Surface): pygame display surface

            cam\_pos (pygame.math.Vector2): position of the camera

        """

        # Draw all tiles

        for sprite in self.tileGroup:

            sprite.draw(screen, cam\_pos)

        # Draw deadzone (can be removed later)

        for dz in self.dead\_zone:

            pygame.draw.rect(screen,YELLOW, pygame.Rect(dz[0], dz[1], dz[2]-cam\_pos.x,dz[3] - cam\_pos.y))

        # Draw checkpoints

        for respawn in self.checkpoint\_group:

            respawn.draw(screen, cam\_pos)

        for tiles in self.disappear\_tiles:

            tiles.draw(screen, cam\_pos)

        self.finish\_point.draw(screen, cam\_pos)

class MenuButton(Button):

    """

    Menu Button

    """

    # Initialise the button

    def \_\_init\_\_(self):

        super().\_\_init\_\_(40,40,(SIZE[0]- MENU\_BUTTON\_PADDING - 40, MENU\_BUTTON\_PADDING),"images/settingButton.png", SCREENTOGAMEMENU)

class GameMenuScreen():

    """

    Game Menu Screen

    """

    def \_\_init\_\_(self):

        self.background = Background(706, 381, (47, 48), YELLOW)

        self.closeButton = CloseButton(24,24, (741, 36), RETURNTOGAME)

        self.quitButton = QuitButton((331, 306))

        self.controlButton = InstructionButton((331,118))

        self.restartButton = RestartButton((331, 212))

        self.gameMenu\_sprite\_group = [self.background, self.closeButton, self.quitButton,self.controlButton, self.controlButton, self.restartButton]

    def drawScreen(self, screen: pygame.Surface):

        """

        Draw Screen

        Args:

            screen (pygame.Surface): pygame display surface

        """

        for sprite in self.gameMenu\_sprite\_group:

            sprite.draw(screen)

    def mouseInteraction(self,position: tuple, status: list, click) -> list:

        """

        Interaction with mouse pointer

        Args:

            position (tuple): position of the pointer

            status (list): status code

        Returns:

            list: status code

        """

        for sprite in self.gameMenu\_sprite\_group:

            status = sprite.mouseInteraction(position, status, click)

        return status

    def keyResponse(self,event: pygame.event, status: list) -> list:

        """

        Response to keys

        Args:

            event (pygame.event): pygame event

            status (list): status codes

        Returns:

            list: status codes

        """

        for sprite in self.gameMenu\_sprite\_group:

            status = sprite.keyResponse(event, status)

        return status

class Enemy(pygame.sprite.Sprite):

    def \_\_init\_\_(self, position: list, x\_boundary: int, y\_boundary: int, imgName: str, size: tuple, hp: int):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load(imgName), size)

        self.size = size

        self.rect = self.image.get\_rect()

        self.rect.x = position[0]

        self.rect.y = position[1]

        self.x\_boundary = x\_boundary

        self.y\_boundary = y\_boundary

        self.x\_speed = 1

        self.hp = hp

        self.max\_hp = self.hp

        self.h\_bar\_height = round(self.rect.height\*0.1)

    def change\_hp(self, value: int):

        """

        Change the HP of the enemy

        Args:

            value (int): the amount of change in HP

        """

        self.hp += value

    def update(self):

        """

        Update the position and speed of the enemy

        """

        self.rect.x += self.x\_speed

        if self.x\_speed >0 and self.rect.x > self.x\_boundary[1]-self.size[0]:

            self.x\_speed \*= -1

        elif self.x\_speed < 0 and self.rect.x < self.x\_boundary[0]:

            self.x\_speed \*= -1

    def draw(self, screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        """

        Draw the enemy on the screen

        Args:

            screen (pygame.Surface): pygame display surface

            cam\_position (pygame.math.Vector2): the position of the camera

        """

        screen.blit(self.image, (self.rect.x - cam\_position.x, self.rect.y))

        pygame.draw.rect(screen, RED, (self.rect.left - cam\_position.x, round(self.rect.bottom - self.rect.height\*0.1), self.rect.width, self.h\_bar\_height))

        pygame.draw.rect(screen, LIGHTGREEN, (self.rect.left - cam\_position.x, round(self.rect.bottom - self.rect.height\*0.1), (self.rect.width \* self.hp/self.max\_hp), self.h\_bar\_height))

class Goomba(Enemy):

    def \_\_init\_\_(self, position, x\_boundary, y\_boundary):

        super().\_\_init\_\_(position, x\_boundary, y\_boundary, os.path.join("images","goomba.png"), GOOMBA\_SIZE, 30)

class Shooter(Enemy):

    def \_\_init\_\_(self, position):

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images","goomba.png")), BLOCKSIZE)

        # self.image = pygame.surface(BLOCKSIZE)

        self.rect = self.image.get\_rect()

        self.position = pygame.math.Vector2(position)

        self.rect.x = self.position.x

        self.rect.y = self.position.y

        self.player\_pos = pygame.math.Vector2(0,0)

        self.bullet\_group = pygame.sprite.Group()

    def load\_player\_pos(self, pos):

        self.player\_pos.x = pos[0]

        self.player\_pos.y = pos[1]

    def shoot(self):

        if self.position.distance\_to(self.player\_pos) < 300:

            pass

class EnemyBullet(GameObject):

    def \_\_init\_\_(self, direction: pygame.math.Vector2, position: tuple):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images", "redRect.png")), (10,10))

        self.rect = self.image.get\_rect()

        self.direction = direction

        self.rect.x = position[0]

        self.rect.y = position[1]

class DeathText():

    def \_\_init\_\_(self):

        big\_font = pygame.font.Font("freesansbold.ttf",100)

        self.txt = big\_font.render("Defeated", True, DARKBLUE)

        fontSize = self.txt.get\_size()

        self.txt\_pos = (182, 130)

    def draw(self,screen: pygame.Surface):

        screen.blit(self.txt, self.txt\_pos)

class GameBoard():

    def \_\_init\_\_(self, death\_count=0, time\_count = 0):

        self.time = 0

        self.death\_image = pygame.transform.scale(pygame.image.load(os.path.join("images", "death.png")), (40,40))

        self.death\_image\_coor = (498,20)

        self.death\_count = death\_count

        self.small\_font = pygame.font.Font("freesansbold.ttf", 30)

        self.start\_time = pygame.time.get\_ticks()

        self.time\_count = time\_count

    def logic(self):

        self.time += 1

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        screen.blit(self.death\_image, self.death\_image\_coor)

        txt = self.small\_font.render(str(self.death\_count), True, BLACK)

        screen.blit(txt, (558, 30))

        time = self.small\_font.render(str((pygame.time.get\_ticks() - self.start\_time)//1000 + self.time\_count), True, BLACK)

        screen.blit(time, (20,20))

class WinText():

    def \_\_init\_\_(self):

        big\_font = pygame.font.Font("freesansbold.ttf",100)

        self.txt = big\_font.render("Victory", True, DARKBLUE)

        self.txt\_pos = (182, 130)

    def set\_time\_death(self, time, death):

        small\_font = pygame.font.Font("freesansbold.ttf",40)

        self.time\_txt = small\_font.render("Time used: "+str(time) + "  Death count: "+ str(death), True, DARKBLUE)

        self.time\_pos = (182+ (self.txt.get\_size()[0] - self.time\_txt.get\_size()[0])/2, 130 + self.txt.get\_size()[1])

    def draw(self,screen: pygame.Surface):

        screen.blit(self.txt, self.txt\_pos)

        screen.blit(self.time\_txt, self.time\_pos)

#### Helper.py

import csv

import logging

import os

from abc import ABC, abstractmethod

from asyncio.windows\_events import NULL

from pyclbr import Function

import pygame

from constants import \*

class Button(pygame.sprite.Sprite, ABC):

    """

    Button Class with mouse interaction and key response method

    """

    # Initialise Button as a sprite object

    def \_\_init\_\_(self,width: int, height: int, position: tuple, imageName: str, status\_code):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.image = pygame.transform.scale(pygame.image.load(imageName), (width, height))

        self.rect = self.image.get\_rect()

        self.rect.x = position[0]

        self.rect.y = position[1]

        self.clicked = False

        self.status\_code = status\_code

    def mouseInteraction(self,position: tuple, status: list, click):

        if click and self.rect.collidepoint(position) and not self.clicked:

            status.extend([self.status\_code])

            self.clicked = True

        elif not click:

            self.clicked = False

        return status

    def keyResponse(self,event: pygame.event, status: list):

        return status

    def draw(self,screen: pygame.Surface, cam\_pos = pygame.math.Vector2(0,0)):

        screen.blit(self.image, (self.rect.x,self.rect.y))

    def update(self):

        pass

class SettingButton(Button):

    def \_\_init\_\_(self, width: int, height: int, position: tuple):

        super().\_\_init\_\_(width, height,position,os.path.join("images","settingButton.png"), SCREENTOSETTING)

class GameObject(pygame.sprite.Sprite, ABC):

    def \_\_init\_\_(self):

        pygame.sprite.Sprite.\_\_init\_\_(self)

    @abstractmethod

    def update(self):

        pass

    @abstractmethod

    def draw(self):

        pass

class WordButton(ABC):

    def \_\_init\_\_(self,width: int, height: int, position: tuple, color: tuple, textColor: tuple, txt: str, status\_code):

        super().\_\_init\_\_()

        self.rect = pygame.Rect(\*position, width, height)

        self.color = color

        font = pygame.font.Font("freesansbold.ttf", FONTSIZE)

        self.txt = font.render(txt, True, textColor)

        fontSize = self.txt.get\_size()

        self.txt\_pos = (position[0] + (width - fontSize[0])/2, position[1] + (height - fontSize[1])/2)

        self.status\_code = status\_code

        self.clicked = False

        self.covered = False

        self.covered\_color = (color[0] \* 0.6, color[1] \* 0.6, color[2] \* 0.6)

    def mouseInteraction(self,position: tuple, status: list, click):

        self.covered = self.rect.collidepoint(position)

        if click and self.covered and not self.clicked:

            status.extend([self.status\_code])

            self.clicked = True

        elif not click:

            self.clicked = False

        return status

    def keyResponse(self,event: pygame.event,status: list):

        return status

    def draw(self, screen: pygame.Surface):

        if self.covered:

            pygame.draw.rect(screen, self.covered\_color, self.rect, 0 )

        else:

            pygame.draw.rect(screen, self.color, self.rect, 0)

        screen.blit(self.txt, self.txt\_pos)

class Background(pygame.sprite.Sprite):

    def \_\_init\_\_(self, width: int, height: int, position: tuple, color: tuple):

        super().\_\_init\_\_()

        self.width = width

        self.height = height

        self.color = color

        self.rect = pygame.Rect(position[0], position[1], width, height)

    def draw(self,screen: pygame.Surface):

        pygame.draw.rect(screen, self.color, self.rect)

    def mouseInteraction(self, position: tuple, status: list, click: bool):

        return status

    def keyResponse(self,event: pygame.event,status: list):

        return status

class CloseButton(Button):

    def \_\_init\_\_(self,width: int, height: int, position: tuple, statusCode: str):

        super().\_\_init\_\_(width, height, position, os.path.join("images","closeButton.jpg"), statusCode)

class QuitButton(WordButton):

    def \_\_init\_\_(self, position: tuple):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Exit Game", CLOSEGAME)

class InstructionButton(WordButton):

    def \_\_init\_\_(self, position: tuple):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Instructions", SCREENTOINSTRUCTION)

class RankScreenButton(WordButton):

    def \_\_init\_\_(self, position: tuple):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Rankings", SCREENTORANKING)

class QuitGameButton(WordButton):

    def \_\_init\_\_(self, position: tuple):

        super().\_\_init\_\_(140, 40, position, (51, 51, 204), WHITE, "Quit Game", EXITGAME)

class Tile(pygame.sprite.Sprite):

    def \_\_init\_\_(self,position: tuple, imgFile: str):

        super().\_\_init\_\_()

        self.x = position[0]

        self.y = position[1]

        self.image = pygame.transform.scale(pygame.image.load(imgFile), BLOCKSIZE)

        self.rect = self.image.get\_rect()

        self.rect.x = self.x

        self.rect.y = self.y

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        screen.blit(self.image, (self.rect.x - cam\_pos.x, self.rect.y))

    def player\_interaction(self, \*\*args):

        pass

class Ground(Tile):

    def \_\_init\_\_(self,position: tuple):

        super().\_\_init\_\_(position, os.path.join("images","groundTile.png"))

class FakeSpike(Ground):

    def \_\_init\_\_(self,position: tuple):

        super().\_\_init\_\_(position)

        self.x = position[0]

        self.y = position[1]

        self.base = self.height = 30

        self.rect\_group = []

        self.createRect()

    def createRect(self):

        base\_change = self.base / 40

        height\_change = self.height / 20

        base\_length = self.base - base\_change\*2

        left = self.x + base\_change

        top = self.y + self.height-height\_change

        for i in range(19):

            rect = pygame.Rect(left, top, base\_length, height\_change)

            self.rect\_group.append(rect)

            base\_length -= base\_change\*2

            left += base\_change

            top -= height\_change

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        screen.blit(self.image, (self.rect.x - cam\_position.x, self.rect.y))

        for rect in self.rect\_group:

            pygame.draw.rect(screen,SILVER, pygame.Rect(rect.left - cam\_position.x, rect.top - cam\_position.y, rect.width, rect.height))

class AirTile(Tile):

    def \_\_init\_\_(self,position: tuple):

        super().\_\_init\_\_(position,os.path.join("images","airTile.png"))

class InstructionScreen():

    def \_\_init\_\_(self, closeStatusCode: str):

        # Instructions

        self.instructions = [

            " 'A' key for going to the left ",

            " 'D' key for going to the right",

            " 'W' key for jumping",

            " 'SPACE' key for shooting"

        ]

        self.fonts = []

        # for generation class

        font = pygame.font.Font("freesansbold.ttf", FONTSIZE)

        # Generate surface object for the instuctions

        start\_y = 108

        for instruction in self.instructions:

            txt = font.render(instruction, True, BLACK)

            fontSize = txt.get\_size()

            # It is to center the text

            txt\_pos = (76 + (648 - fontSize[0])/2, start\_y + INSTRUCTION\_MENU\_PADDING)

            # Add the padding for the coordinate of the next text

            start\_y += fontSize[1] + INSTRUCTION\_MENU\_PADDING \* 2

            self.fonts.append((txt,txt\_pos))

        self.background = Background(648,336,(76, 64), SETTINGSCREENCOLOR)

        self.closeButton = CloseButton(24,24, (712, 52), closeStatusCode)

        self.instruction\_sprite\_group = [self.background, self.closeButton]

    def mouseInteraction(self, position: tuple, status: list, click):

        return self.closeButton.mouseInteraction(position, status, click)

    def keyResponse(self,event: pygame.event, status: list):

        return status

    def drawScreen(self, screen: pygame.Surface):

        for sprite in self.instruction\_sprite\_group:

            sprite.draw(screen)

        for txt, txt\_pos in self.fonts:

            screen.blit(txt, txt\_pos)

class RestartButton(WordButton):

    def \_\_init\_\_(self, position: tuple):

        super().\_\_init\_\_(140,40, position, (51, 51, 204), WHITE, "Restart", RESTARTGAME)

def xReToDe(x\_pos: int) -> int:

    """

    Relative x coordinate (Block size) to pygame x coordinate

    Args:

        x\_pos (tuple): relative x coordinate position

    Returns:

        int: the corresponding pygame x coordinate

    """

    return int(x\_pos)\*BLOCKSIZE[0]

def yReToDe(y\_pos: int) -> int:

    """

    Relative y coordinate (Block size) to pygame y coordinate

    Args:

        y\_pos (tuple): relative y coordinate position

    Returns:

        int: the corresponding pygame y coordinate

    """

    return int(y\_pos)\*BLOCKSIZE[1]

def relativeCoor2DeCoor(relativePosition: tuple)-> tuple:

    """

    take in relative coordinate and return pygame coordinate

    Args:

        relativePosition (tuple): relative coordinate (1 = 1 Block Size)

    Returns:

        tuple: pygame coordinate

    """

    return (xReToDe(relativePosition[0]), yReToDe(relativePosition[1]))

def deCoor2RelativeCoor(dePosition: tuple) -> tuple:

    """

    Pygame coordinate to realtive coordinate

    Args:

        dePosition (tuple): pygame coordinate

    Returns:

        tuple: relative coordinate

    """

    return (dePosition[0]/BLOCKSIZE[0], dePosition[1]/BLOCKSIZE[1])

def apply(ls: list, func: Function, \*\*args) -> list:

    """

    Apply a function to items in a list with optional arguments

    Args:

        ls (list): list to be applying to

        func (Function): function to apply

        \*\*args: list of arguments to be passed to

    Returns:

        list: The list with function applied to each element

    """

    result = []

    argument = ""

    for key, value in args.items():

        argument = argument + ", " + str(key) +"="+str(value)

    argument = argument + ")"

    for i in ls:

        com = f"x = {func.\_\_name\_\_}({ i }" + argument

        loc = {}

        exec(f"{com}",globals(),loc)

        x = loc["x"]

        result.append(x)

    return result

# Level status code

def create\_level\_status\_code(level: int):

    return f"Initialise Level {level}"

def check\_status\_init\_level(string: str):

    return bool(re.match("^(Initialise Level)\s\-?[0-9]+$", string))

def extract\_level\_from\_status\_code(status\_code: str):

    if not check\_status\_init\_level(status\_code):

        return None

    try:

        return int(status\_code.split(" ")[-1])

    except ValueError:

        return None

class Spike:

    def \_\_init\_\_(self, position: tuple, base: int, height: int):

        # Position of the square the spike is int

        self.x, self.y = position[0], position[1]

        # The base and height of the triangle

        self.base = base

        self.height = height

        self.rect\_group = []

        self.createRect()

    # The spike will made of 19 rectangle stacking on top of each other, mimicing a isosceles triangle. (The 20th has width of 0, so it will be ignore)

    def createRect(self):

        # The decrease in base for every rectangle on top

        base\_change = self.base / 40

        height\_change = self.height / 20

        # Base of the first triangle

        base\_length = self.base - base\_change\*2

        # The coordinate of the first rectangle in the first iteration

        left = self.x + base\_change

        top = self.y + self.height-height\_change

        for \_ in range(19):

            rect = pygame.Rect(left, top, base\_length, height\_change)

            self.rect\_group.append(rect)

            base\_length -= base\_change\*2

            left += base\_change

            top -= height\_change

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        for rect in self.rect\_group:

            pygame.draw.rect(screen,SILVER, pygame.Rect(rect.left - cam\_position.x, rect.top - cam\_position.y, rect.width, rect.height))

    def logic(self):

        pass

    def player\_interaction(self,player\_rect):

        pass

class ActivateObjects(ABC):

    def \_\_init\_\_(self,zone: list):

        self.zone = pygame.Rect(\*zone)

        self.activate = False

    def detect(self, player\_rect: pygame.Rect):

        if player\_rect.colliderect(self.zone):

            self.activate = True

    def logic(self):

        pass

    @abstractmethod

    def player\_interaction(self,player\_rect):

        pass

class SpikeUp(Spike, ActivateObjects):

    def \_\_init\_\_(self, position: tuple, base: int, height: int, zone: list, up: int, hori\_dir = 0): #zone = (leftx, rightx, width, height)

        Spike.\_\_init\_\_(self, (position[0], position[1]), base, height)

        ActivateObjects.\_\_init\_\_(self,zone)

        self.tar\_y = position[1] - up\*BLOCKSIZE[1]

        # Number of block upward it should moves

        self.vert\_dir = up

        # Number of block to the right it should moves

        self.hori\_dir = hori\_dir

        self.tar\_x = position[0] + hori\_dir \* BLOCKSIZE[0]

    # Return if the player collides with the spike

    def player\_interaction(self, player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

        if self.activate and player\_rect.collidelist(self.rect\_group) != -1:

            return True

        return False

    # Move the spike if it is activated and it has not reached its final position

    def logic(self):

        if self.activate:

            # Move up

            if self.y > self.tar\_y and self.vert\_dir > 0:

                self.y -= UP\_SPEED

                for rect in self.rect\_group:

                    rect.y -= UP\_SPEED

            # Move down

            elif self.y < self.tar\_y and self.vert\_dir < 0:

                self.y += UP\_SPEED

                for rect in self.rect\_group:

                    rect.y += UP\_SPEED

            # Move right

            if self.x < self.tar\_x and self.hori\_dir > 0:

                self.x += HORI\_SPEED

                for rect in self.rect\_group:

                    rect.x += HORI\_SPEED

            # Move left

            elif self.x > self.tar\_x and self.hori\_dir < 0:

                self.x -= HORI\_SPEED

                for rect in self.rect\_group:

                    rect.x -= HORI\_SPEED

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        # pygame.draw.rect(screen, YELLOW, pygame.Rect(self.zone.left - cam\_position.x, self.zone.top-cam\_position.y, self.zone.width, self.zone.height))

        super().draw(screen, cam\_position)

class Appear\_block(ActivateObjects, Ground):

    def \_\_init\_\_(self,position: tuple):

        ActivateObjects.\_\_init\_\_(self, (position[0]-10, position[1]-10, BLOCKSIZE[0]+20, BLOCKSIZE[1]+20))

        Ground.\_\_init\_\_(self,position)

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        if self.activate:

            screen.blit(self.image, (self.rect.x - cam\_pos.x, self.rect.y))

    def player\_interaction(self,player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

class DisappearBlock(ActivateObjects, Ground):

    def \_\_init\_\_(self,position: tuple):

        # easier

        # ActivateObjects.\_\_init\_\_(self, (position[0]- 3, position[1]- 3, BLOCKSIZE[0] + 6, BLOCKSIZE[1] + 6))

        ActivateObjects.\_\_init\_\_(self, (position[0], position[1], BLOCKSIZE[0], BLOCKSIZE[1]))

        Ground.\_\_init\_\_(self,position)

        self.rect = NULL

        self.position = position

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        if not self.activate:

            screen.blit(self.image, (self.position[0] - cam\_pos.x, self.position[1]))

    def player\_interaction(self,player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

class GrowSpike(Spike, ActivateObjects):

    def \_\_init\_\_(self, position: tuple, base: int, height: int, zone: list, up: int, hori: int):

        Spike.\_\_init\_\_(self, position, base, height)

        ActivateObjects.\_\_init\_\_(self,zone)

        self.tar\_height = height\*up

        self.tar\_base = hori \* base

    def player\_interaction(self, player\_rect: pygame.Rect):

        if not self.activate:

            self.detect(player\_rect)

        if self.activate and player\_rect.collidelist(self.rect\_group) != -1:

            return True

        return False

    def logic(self):

        flag = False

        if self.activate and self.height < self.tar\_height:

            self.height += GROW\_SPEED

            self.y -= GROW\_SPEED

            flag = True

        elif self.activate and self.base < self.tar\_base:

            self.base += GROW\_SPEED

            self.x -= GROW\_SPEED /2

            flag = True

        if flag:

            self.rect\_group = []

            base\_change = self.base / 40

            height\_change = self.height / 20

            base\_length = self.base - base\_change\*2

            left = self.x + base\_change

            top = self.y + self.height-height\_change

            for i in range(19):

                rect = pygame.Rect(left, top, base\_length, height\_change)

                self.rect\_group.append(rect)

                base\_length -= base\_change\*2

                left += base\_change

                top -= height\_change

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        #pygame.draw.rect(screen, YELLOW, pygame.Rect(self.zone.left - cam\_position.x, self.zone.top-cam\_position.y, self.zone.width, self.zone.height))

        super().draw(screen, cam\_position)

class Check\_point(Tile, ActivateObjects):

    def \_\_init\_\_(self, position: tuple):

        Tile.\_\_init\_\_(self, position, os.path.join("images","respawn\_before.png"))

        ActivateObjects.\_\_init\_\_(self, (position[0]-10, position[1]-10, BLOCKSIZE[0] + 20, BLOCKSIZE[1] + 20))

    def player\_interaction(self,player\_rect: pygame.Rect):

        if player\_rect.colliderect(self.zone):

            self.image = pygame.transform.scale(pygame.image.load(os.path.join("images","respawn.png")), BLOCKSIZE)

            return [self.x, self.y +18]

class HorizontalSpike():

    def \_\_init\_\_(self, position: tuple, base: int, height: int):

        self.x = position[0]

        self.y = position[1]

        self.base = base

        self.height = height

        self.rect\_group = []

        self.createRect()

    def createRect(self):

        base\_change = self.base / 40

        height\_change = self.height / 20

        base\_length = self.base - base\_change\*2

        left = self.x

        top = self.y + height\_change

        for i in range(19):

            rect = pygame.Rect(left, top, height\_change, base\_length)

            self.rect\_group.append(rect)

            base\_length -= base\_change\*2

            left += height\_change

            top += base\_change

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        for rect in self.rect\_group:

            pygame.draw.rect(screen,SILVER, pygame.Rect(rect.left - cam\_position.x, rect.top - cam\_position.y, rect.width, rect.height))

    def logic(self):

        pass

    def player\_interaction(self,player\_rect):

        pass

class MoveableHoriSpike(HorizontalSpike, SpikeUp):

    def \_\_init\_\_(self, position: tuple, base: int, height: int, zone: list, up: int, hori\_dir = 0):

        SpikeUp.\_\_init\_\_(self, position, base, height, zone, up, hori\_dir)

        HorizontalSpike.\_\_init\_\_(self,position, base, height)

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        #pygame.draw.rect(screen, YELLOW, pygame.Rect(self.zone.left - cam\_position.x, self.zone.top-cam\_position.y, self.zone.width, self.zone.height))

        HorizontalSpike.draw(self,screen, cam\_position)

    def player\_interaction(self,player\_rect: pygame.Rect):

        SpikeUp.player\_interaction(self,player\_rect)

    def logic(self):

        SpikeUp.logic(self)

class Bullet(GameObject):

    def \_\_init\_\_(self, direction: int, position: tuple):

        super().\_\_init\_\_()

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images", "redRect.png")), BULLET\_SIZE)

        self.rect = self.image.get\_rect()

        self.direction = direction

        self.rect.x = position[0]

        self.rect.y = position[1]

    def update(self, tiles: list, x\_boundary):

        self.rect.x += self.direction \* BULLET\_SPEED

        if len(pygame.sprite.spritecollide(self, tiles, False)) != 0 or self.rect.right < x\_boundary[0] or self.rect.left > x\_boundary[1]:

            self.kill()

    def draw(self, screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        screen.blit(self.image, (self.rect.x - cam\_position.x, self.rect.y))

class FinishPoint(Tile):

    def \_\_init\_\_(self, position: tuple):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.image = pygame.transform.scale(pygame.image.load(os.path.join("images", "flag\_red.png")), (BLOCKSIZE[0], BLOCKSIZE[1]\*5))

        self.rect = self.image.get\_rect()

        self.rect.x = position[0]

        self.rect.y = position[1]

    def player\_interaction(self,player\_rect):

        pass

    def draw(self, screen: pygame.Surface, cam\_pos: pygame.math.Vector2):

        screen.blit(self.image, (self.rect.x - cam\_pos.x, self.rect.y))

class TrapGroup():

    def \_\_init\_\_(self, filename: str):

        self.all\_trap\_group = []

        self.parseFile(filename)

    def parseFile(self,filename: str):

        # Maybe not use standard csv file but determine how to read the content by the first column

        with open(filename, 'r') as f:

            reader =  csv.reader(f)

            next(reader)

            for row in reader:

                relPos = (int(row[1]), int(row[2]))

                dePos = relativeCoor2DeCoor(relPos)

                if row[0] == "normal\_spike":

                    for i in range(int(row[5])):

                        for j in range(int(row[6])):

                            trap = Spike(relativeCoor2DeCoor((int(row[1])+ i, int(row[2])+j)), int(row[3]), int(row[4]))

                            self.all\_trap\_group.append(trap)

                elif row[0] == "up\_spike":

                    trap = SpikeUp(dePos, int(row[3]), int(row[4]),apply(row[5:9], int), int(row[11]), int(row[12]))

                    self.all\_trap\_group.append(trap)

                elif row[0] == "grow\_spike":

                    trap = GrowSpike(dePos, int(row[3]), int(row[4]),apply(row[5:9], int), int(row[11]), int(row[12]))

                    self.all\_trap\_group.append(trap)

                elif row[0] == "hori\_spike":

                    trap = HorizontalSpike(dePos, int(row[3]), int(row[4]))

                    self.all\_trap\_group.append(trap)

                elif row[0] == "hori\_move\_spike":

                    trap = MoveableHoriSpike(dePos, int(row[3]), int(row[4]),apply(row[5:9], int), int(row[11]), int(row[12]))

                    self.all\_trap\_group.append(trap)

    def draw(self,screen: pygame.Surface, cam\_position: pygame.math.Vector2):

        for trap in self.all\_trap\_group:

            trap.draw(screen, cam\_position)

    def logic(self):

        for trap in self.all\_trap\_group:

            trap.logic()

    def player\_interaction(self, player\_rect: pygame.Rect):

        for trap in self.all\_trap\_group:

            trap.player\_interaction(player\_rect = player\_rect)

#### constant.py

# Color

BROWN = (135, 52, 35) #873423

SKYBLUE = (91, 148, 251) #5B94FB

LIGHTGREY = (213, 216, 220)

BLACK = (0, 0, 0)

YELLOW = (251,208,0)

WHITE = (255,255,255)

GOLD = (212, 175, 55)

DARKBLUE = (0, 0, 139)

SETTINGSCREENCOLOR = (144, 212, 204)

SILVER = (192, 192, 192)

RED = (255,0,0)

LIGHTGREEN = (144,238,144)

#Size

SIZE = (810,540)

BLOCKSIZE = (30,30)

SETTINGS\_BUTTON\_SIZE = (105, 40)

LEVEL\_BUTTON\_SIZE = (148, 40)

LEVEL\_IMAGE\_SIZE = (240, 168)

PLAYER\_SIZE = (int(BLOCKSIZE[0]\*0.4), int(BLOCKSIZE[1]\*0.4))

FONTSIZE = 20

# Padding

LEVEL\_IMAGE\_BUTTON\_PADDING = EDGE\_LEVEL\_IMAGE\_PADDING = 16

LEVEL\_2\_LEVEL\_PADDING = 24

INSTRUCTION\_MENU\_PADDING = 10

#Misc

EXIT = "exit"

NUM\_OF\_LEVELS = 2

#Screens

MENU = "menu"

SETTINGSCREEN = "setting screen"

GAMESCREEN = "game screen"

INSTRUCTIONSCREEN = "instruction screen"

RANKINGSCREEN = "ranking screen"

GAMEMENU = "game menu"

GAME = "game"

DEATHSCREEN = "death screen"

WINSCREEN = "win screen"

#Status Code

SCREENTOSETTING = "change to setting screen"

SCREENTOGAMEMENU = "change to game menu screen"

RETURNTOGAME = "close game menu screen"

CLOSEGAME = "close game"

EXITGAME = "exit game"

SCREENTOINSTRUCTION = "change to instruction screen"

SCREENTORANKING = "change to ranking screen"

RESTARTGAME = "restart game"

PLAYERDEATH = "player death"

PLAYERWIN = "player wins"

# Camera

PLAYER\_LEFT\_PADDING =  150

PLAYER\_RIGHT\_PADDING = SIZE[1] - 100

# Player

PLAYERSTARTPOS = (PLAYER\_LEFT\_PADDING,BLOCKSIZE[1] \* 7 )

HORIZONAL\_MAX\_SPEED = BLOCKSIZE[1] \* 0.15

JUMP\_SPEED = BLOCKSIZE[1] / 2

GRAVITY = BLOCKSIZE[1] /25

RIGHT = "right"

LEFT = "left"

#Player statusCode

MOVING\_LEFT = "moving left"

MOVING\_RIGHT = "moving right"

# Game

MENU\_BUTTON\_PADDING = 20

#Enemy

GOOMBA\_SIZE = (BLOCKSIZE[0], BLOCKSIZE[1])

# Spike

UP\_SPEED = 10

GROW\_SPEED = 10

HORI\_SPEED = 10

BULLET\_SIZE = (round(BLOCKSIZE[0]\*0.2), round(BLOCKSIZE[1]\*0.1))

BULLET\_SPEED = 8

BULLET\_TIMER = 5

BULLET\_DAMAGE = 10

ENEMY\_BULLET\_SIZE = (round(BLOCKSIZE[0]\*0.2), round(BLOCKSIZE[1]\*0.2))

BULLET\_SPEED = 5

#### Camera.py

import pygame

from constants import \*

vec = pygame.math.Vector2

class Camera:

    def \_\_init\_\_(self,player, vertices: list):

        # The edges of the camera

        self.vertices = vertices

        self.player = player

        # The position of the camera

        self.position = vec(0,0)

        self.screenW, self.screenH = SIZE

        self.left\_padding = PLAYER\_LEFT\_PADDING

        self.right\_padding = PLAYER\_RIGHT\_PADDING

        self.position.x = self.player.rect.x - self.left\_padding

        self.position.x = max(self.vertices[0], self.position.x)

        self.position.x = min(self.vertices[1] - self.screenW, self.position.x)

        self.position.x = int(self.position.x)

    def scroll(self):

        # If the player coordinate is smaller than the left padding relative to the camera

        if self.player.rect.x < self.position.x + self.left\_padding:

            # Set the position fo the camera so that the player is on the left padding unless the camera reaches the edge of the level

            self.position.x = self.player.rect.x - self.left\_padding

            self.position.x = max(self.vertices[0], self.position.x)

        # If the player coordinate is larger than the right padding relative to the camera

        elif self.player.rect.x > self.position.x + self.screenW - self.right\_padding:

            # Set the position fo the camera so that the player is on the right padding unless the camera reaches the edge of the level

            self.position.x = self.player.rect.x + self.right\_padding - self.screenW

            self.position.x = min(self.vertices[1] - self.screenW, self.position.x)

# Evaluation

## Success Criteria Evaluation

(reference to the testing)