# Introduction

## Project Background and Motivation

When a foul is committed in the game of snooker, the player who the foul was committed against can request for the balls to be replaced to their positions before the foul shot and have their opponent play the shot again.

If a referee is present, it is their job to replace the balls, and in televised matches the TV cameras are rewound to inspect their previous positions if necessary. Due to the inaccuracy of this method, players sometimes disagree with how the referee has placed the balls, and there have been instances of players disagreeing with each other on this matter.

In non-televised matches, be it non-TV tables at professional tournaments, amateur tournaments, or casual settings, there is obviously no TV camera to rewind. While some snooker clubs may have cameras recording the tables, these are usually not configured in a way where their footage could be accessed while a game is ongoing. This presents the issue that, for all the listed scenarios, there is no solution at all for accurately replacing the balls – only the player’s judgement of the positions can be used as a resource, and particularly if more than one balls needs replacing, this is likely to be unreliable and inaccurate.

Given the understanding we now have of this unsolved issue in the game, the aim of this project is to create a software solution which greatly eases this process, and, importantly, removes subjectivity from the matter.

## Scope

While all planned functionality is listed below as within scope, obviously a comprehensive list of everything out of scope cannot be produced. The functionality specified as outside of scope is a clarification on ideas that were mentioned or potentially implied in the Project Specification, Abstract, or Introduction.

Within scope:

* Tracking balls and their colours
* Supporting video throughput
* Recalling a previous position

Outside of scope:

* Advanced output methods such as augmented reality
* Automatic scoring
* Automatically detecting foul shots, or any other referee-called decision

## Project Aims

1. Detect the positions of the snooker balls in an image of a snooker table.
   1. Also detect the colour of each ball.
2. Display the tracked information in some way that can be used to restore the balls to this state.
3. Support video throughput as a proof-of-concept for a camera feed.
   1. Store each new shot as it is played.
   2. Allow displaying a previous table state on-demand.
4. Explore advanced output methods such as augmented reality.

# Research

## Similar Projects

As part of the research stage of the project, projects that aim to tackle similar problems or are technically relevant were identified and their technologies and approaches were reviewed. The information gathered was used to inform decisions on technologies and design approach.

None of the projects found were markedly complete at the time of research, or writing, and their working status is unknown. For this reason, they cannot be analysed as products but simply as projects. However, this does not detract from the fact that they have all implemented a significant amount of core functionality and much can be learned from them.

### The Pool Ball Tracker (Marr, 2012)

The team at GoCardless worked on this system over a 48-hour ‘Hackathon’ and their experience is summarised at high level in a follow-up blog post.

They developed their system in C, using OpenCV. They convert each input frame from an overhead webcam to HSV and mask the image multiple times using each of the colours of the balls.

After initially using the Hough transform (via OpenCV’s *houghCircles* function) to detect balls in the filtered image, they found greater reliability using *findContours*, an OpenCV function which detects generic shapes in an image. This was due to motion blur and slow shutter speed distorting the balls when they were in motion, leading to *houghCircles* failing to detect them.

Marr notes towards the end of the post that the system works at a low 10 frames per second, due to expensive computation. This can be taken as a fair warning to be careful about the algorithms and approaches used in design, as they could quickly become very costly.

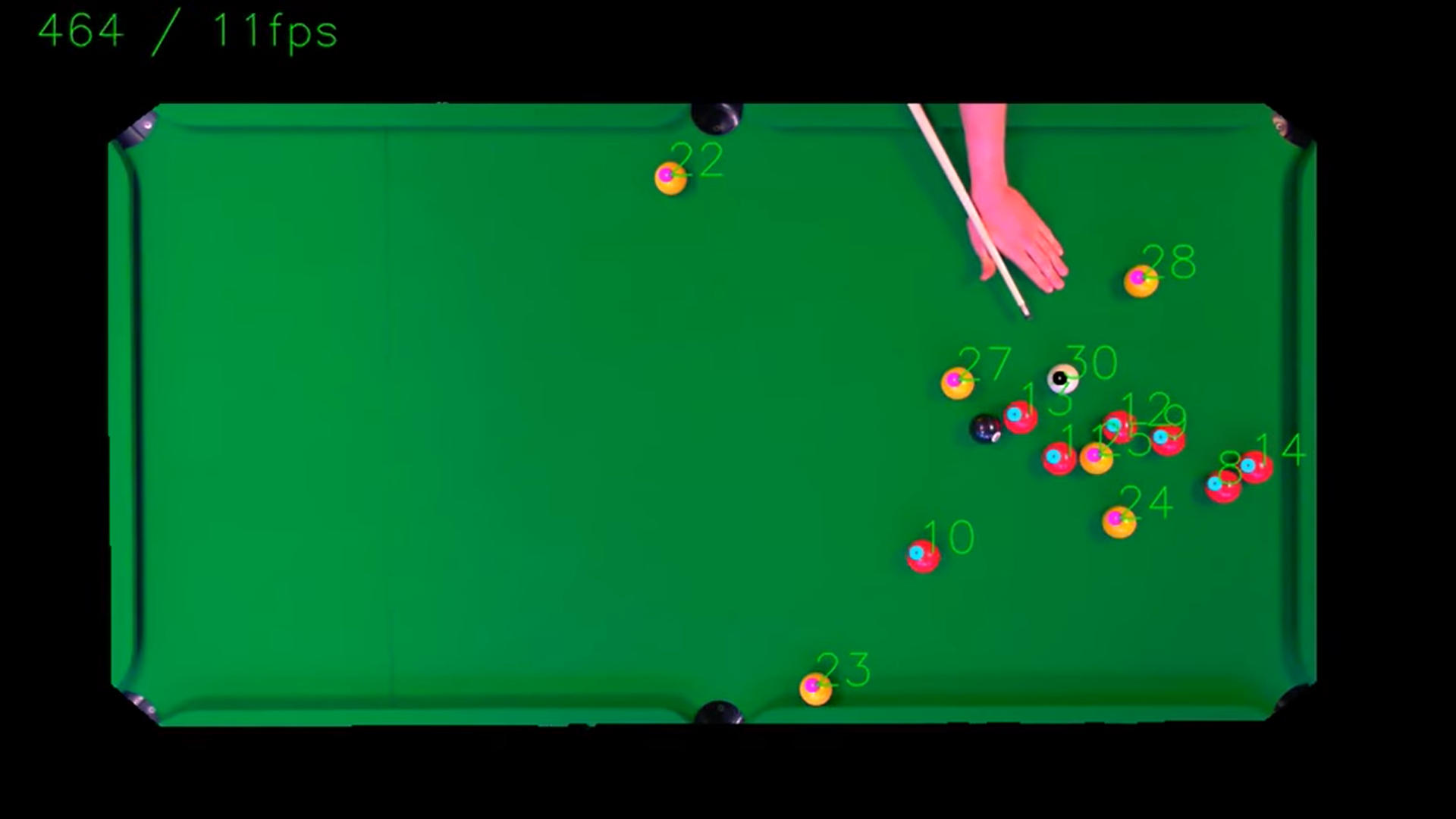


Figure 1: The Pool Ball Tracker's output as of the end of the ‘Hackathon’.

As shown in Figure 1, the ball tracking itself seems to be able to detect all the balls and does not produce any false positives, giving merit to their approach.

### Snooker balls tracking on video (Trusov, 2022)

This notebook goes into detail about the technical approach and outlines the power of OpenCV when it comes to image processing.

The system is developed in Python, using OpenCV, and relies on template matching for the core functionality of ball detection. This means that the solution may need new templates manually creating if a different camera or camera setup was used, unless convincing perspective correction were to be implemented.

Trusov performs pattern matching in a greyscale colour space for the sake of speed, and this has the interesting effect of grouping the patterns based on lightness value, making determining colours more straightforward as they are automatically partially sorted past the pattern matching phase.

The code runs slower than real-time with the 2:42 duration clip linked at the bottom of the notebook taking 9:47 to be processed – another early warning about performance of these types of computer vision applications.

![A picture containing text, pool table, indoor, poolroom

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDgRXhpZgAATU0AKgAAAAgABAE7AAIAAAAHAAAISodpAAQAAAABAAAIUpydAAEAAAAOAAAQyuocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGlwYXNoYQAAAAWQAwACAAAAFAAAEKCQBAACAAAAFAAAELSSkQACAAAAAzU2AACSkgACAAAAAzU2AADqHAAHAAAIDAAACJQAAAAAHOoAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyMDIyOjA0OjAzIDE1OjQ0OjE4ADIwMjI6MDQ6MDMgMTU6NDQ6MTgAAABpAHAAYQBzAGgAYQAAAP/hCxlodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDIyLTA0LTAzVDE1OjQ0OjE4LjU1OTwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5pcGFzaGE8L3JkZjpsaT48L3JkZjpTZXE+DQoJCQk8L2RjOmNyZWF0b3I+PC9yZGY6RGVzY3JpcHRpb24+PC9yZGY6UkRGPjwveDp4bXBtZXRhPg0KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICA8P3hwYWNrZXQgZW5kPSd3Jz8+/9sAQwAHBQUGBQQHBgUGCAcHCAoRCwoJCQoVDxAMERgVGhkYFRgXGx4nIRsdJR0XGCIuIiUoKSssKxogLzMvKjInKisq/9sAQwEHCAgKCQoUCwsUKhwYHCoqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioq/8AAEQgCLAPlAwEiAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A8yCkU8UUzNZFWH0UzdS7qAsOooFKaACjNJSE4pD1HZopoPFJmmGo6gUgPNLmkIXNN3Cmk8000BYfvFNJphNNLYpiJM0ZqLeaN9AEmaXJ9agMmKTzqAJ93vSFveq3m+9HmUATbuaN4qDzKTfQBY30hfNVy9AkoAsZpN2O9Q+ZSb6YFjzKQvUG+kLZNICffSb6g3e9JupgTF6Teai3Um6nYCffSb8VFmkZsClYCQye9N3+9QF6TfTsBYL+9J5lQFqbvpWAsbs1UvGO0ipUfmo7mPzACKWwEenw+WWmfgAVoRsi2vmbuSx/KoboeVYbE7iqdqTIhQsQQOKhu4Jm+1m6RQ3ER8yJxnI7H0NEgymOhNQ6PqEunttK5U87W6VZuZftEzTkBQxJwO1SkNmNeSFIXQ/erG3HdzWnqEqtIcVnJE80gSJSzHoBWiYWNO3GVGK1La7eCF41+7IMEVQgieI7ZOCOoqeqJsPJFNJopDSAQ0zBJp9HSmAgFGaC1Nzk0CH0A0CkJxTAUkU0sAOahluUTvzVKW7L8CgCzJcqp4qFrvd0qmxLdTUkCjNAFgK0nWpFiK9KdGOlTcbaVxkJU1ExzUssmFwOtVlyTzQInhAHWpxUCHAqQNVASqcU8NVfdTg1AFjNOBqBWp26gZYDCpAwxVQPTw1KwFkP6UokquGIo3miwWLe/NNDc9agV6XdSAmzzTwfeoA9LuoAnB96du461W3Uu+gCbd70m4etQ76TfQBMW96N3vUG+jzKLAWN1G6q++jzKALIal31WD0eZRYCzvoL1W8ykMlICzvFG6qu+nB6ALO6k3e9QFqTfigCxmiq/mUu/wB6AJWNJmoi/vRvpjJw3HWlz71X30b6QixmmFjmot9G6gCXd704N71Bvpd1AywCKMioA1KHpgT5pc5qHfT1akIfRim5NLQMXFJQTSZoGLiigGkzQMXikNBNJQIKKKaTTEBIph5oPNABosA0igU/bTSOKQBRRRQBcNNxTiabQUhopQOaKUUAPopM0E8UALTWpN1LtLUhjc0ma0bDRb7UcfZbeRx6gcVrjwBq7cmIL9TWUq0I7s0VOT6HL5xR5laniPwtqGi6Q95OQFUgcHnmuHa5n/56v/30auEozV0P2T6nQmVfWmNMtc79qmH/AC0b86Ptcv8Afb86on2bOgMqnvTTKvrWF9ql/vn86PtEp/jNUHsmzbMoNN8wetYwnl/vmnefJ/epB7Jo1HkHrTN49azTNIf4qTzX/vUB7Jml5g9aXeD3rN3v/epd7/3qYeyZoF/ejzB61nGR/wC8aPMb+9QHsmaO8etAZfWs/wAxvWnB29aBeyZf3r60bx61Q3P60oZj3pB7Jl3ePWk3+9VcH1pQCe9MXs2Wd49acGHrVXafWnYb1oD2bJmYU3cKj2k96Np9aBezZJuFBbNR7G9aNh9aLh7NimjNJ5ZpfKPrTD2bGE0mal8njrTfJNAezYzdiporhMbXqJoiKqTRuhyM0mLlaNOZ0kjwDUMcYij3Ac1mic4rVsyZrfBqNCAjm3csahu76RcqrfLTp4mQHFZ0iMzc0tBke8u5zzXpNjoWn+HvCK3WrzRpeXYDoq8sq9q4O2ijiw8nBqzfX02oyL5kjOqDAyc0jSMklZlkMryOVJIJyCe9OxVSFitT+Z8taGTHF8UZzURagPTES54pDzTAadQMKKQ9aY8qr1NADy4AOTVC4vzjCUOzztgdKF0/d1oLUGyg0hY5NNLmtM6aKadMHrTuX7JmcGOaswk5qYacA3WtbRNCbVtUhs1k2GT+IjpUSlbUPYtvQoK3FDSHFeo2Xwvsgn+kXLscdhT7r4XWHknyLllb3Fcv1ynex0fUa1r2PJHam5zW54k8NS6DfCB5RJuXcCPSscREDmumM1JXRh7CadmKh4p+6kWM4p3lH1q+ZB7CQganqab5R9aUIRRzIPYTJAaXNNCt604I3rRzB7CYZpwY0mxqXY1HMHsJi7zTg1M2NS4cUcyD2M0SBqduqHD+tGH7Gi6F7GZNupQ1QYf1pNzjvRdC9jMs5pN1Vt0ntRmT1FGg/YyLG6k3VBmSkxLRdB7GRYL0m8VBiT2oxJ7UXF7ORYz70uarZkzSl39KA9nIsZozVbzJOwppmf8Au/rQL2ci1uFG6qZmcfw037S/92kHJIv5o3YqkLlvSg3Jx0oDkkWzJSb6pfaW9KPtRp6B7ORd30bzVA3jelJ9rb0p6B7ORoBzS76z/tbelH2w+lFw9nIvebSiSs/7XR9sI7UtA9nI0d9HmVnfbT6UfbT6Ug9nI0hIM07zKyxfH+7S/b/b9aYvZyNMPTw9ZYvx3X9aX+0R/doDkkagkp4krIGoD0pw1ID+GgOSRsBvenA1kDVgP4KcNV/2KRSpyNegCsoauB/AaX+1/RP1oD2cjUziu78PfCDxV4gtUuhbpZQSLuQ3J2sw+nWj4E6DbeLvGk1xfQ77XSYlmKMMq8jHCg+wwT+Ar6lHHA4FWkS9ND50P7P/AIl/5/bH82/wpD+z/wCJu17Y/m3+FfR2aTcKdkQfOP8Awz/4n/5/bH/vpv8ACj/hn7xP/wA/tj/303+FfR9N3UrDPmi9+Aviy1tzJA9nckDJRHIJ+ma85u7O5069ktL+CS3uIm2vFIuGU19vA5ryT48eE7W98Or4ghjCXtmwR3UcyRnjB9cdqLAfO56Uwin9qaakBtFKetFICyTTaWgDNBQdqTNOI4puKBMcOlBpBRQCEVctXaeC/Co1Zzc3f/HuhGB/eNcdEpeQKOpOBXuuhWosNEtoFULtQE49TXDjazpwst2ddCN3qXoYLezgEcESRovZRiopJ03dRQ0U19eLa2/U8s2Og9K6ex8MW0cLNs8wOMse+enT8a8iFCc1ds65VYQ0PJPied3gqcrz+8X+deGMD6V9SePPh1Ne+H7sWU0ioIN5i6h3U5H0zXz4fC12zfeT6V6uHtRhaTOGvjKcNWcywox7V0h8I3RPLLS/8Ifd/wB5a6PrFPucn9oUP5jmse1KBXSf8Ihd+q0o8IXeOq0vrNLuV/aGH/mOaxS103/CIXfqtJ/wiN3/ALNL6xT7h/aGHf2jm8UmDXTf8Ijeeq0o8I3f+z+dP6xS7i+v4b+Y5nml5rp/+ERu/wDZ/Ok/4Q+8PQr+dH1mn3H9ew/8xzH4UYrpv+EOve5X86cPB956rS+s0l1J+v4f+Y5kCnjNdL/wh136rSjwhdjuKPrNJ9R/X8P/ADHNYorph4Quz/EtL/whtyf+Wij8Kf1mn3H9ew385zYHFLzXSjwddf8APRfypf8AhDrvtIlL6zT7h9dw/wDMc2KdXR/8Ifd/30p3/CG3OOZk/Kj6xT7i+uUP5kc1mjmul/4Q25/57p+VKPB9wP8Alsn5Gn9Ypdx/W6H8yOaFKa6T/hEbj/nsn5Gl/wCESuP+eyflT+sU+5X1uh/MjmsGiul/4RO4/wCe0f5Gj/hErk9JY/yo+sU+4vrlD+dHN80c10n/AAiV1/z0jo/4RK6/56R0e3p/zB9cofzo5k5pCM9RXSnwjdH/AJaJR/wiF3/fSq+sU+4fWqD+0jljDF3QflVm3eOHpwO9bp8H3h53IKgk8JXisOVJJwMVDrU+5CrUpOykiLT7Z9Y1GKxs4vMllYKoAzXV+NfAel+DdBiuLq98y/kXAgC4+b/CvR/h34IsvAPhebxJ4g2m9ePKKf4B2A9zXkvjH+1fF+uyahcDbGTiOMn7q9q5fbKU99AnVpwdpM89cs7ZNWrRMrmtS48M3cceSoIHpUQs5LeJWdCFJIBxwTXfGUWtAjUjNXixEj4pxQYpwpa0GRbKbtqU0wmgQ3GKQuB3qzHp1zdL+5QmrMXhK/n5JVfY1k61Nbsz9rT6sxZLktwlRiN2OWzXUxeCrperr+VTjwfcgcyJU/WKfc1jXoLeSOZiTZ0qwK6AeEroDhkP40h8K3Y67fzpe3h3No4qh/MjBIppFb//AAjF3/s/nR/wjN0Ouz86Pbw7l/WqH8yOfCZNdF4JQL4ss93csP0NIPDd0P7n51t+EdLksvF2mSzAbVuUBx3BOP61MqsJRsmEcVRTupI9p0XwfcXaeZdnyl/u96tah4IdbZ2s5C7qM7T3rtIGG51/2zgZqfoK890lcp4yq5XufIXxLUjXIFIwRFyPTmuLxjtXqnxT0Cafx9feUQUyCo6YyK4tvC16OgU/jXbSqQjHlbCWMpN3k7MwAcdqD0rd/wCEXvv7q/nS/wDCL3391PzrX2lPuH1uj/MjAGc0vPatv/hGb7P3V/Onf8Ivfnoq/nS9rT7j+t0b/EjEXNPBwK1/+EY1AfwL+dKPDOof3F/On7Sn3K+tUf5kY+40hJ9a2v8AhGL/APuL+dIfDOoD+BT+NJVYdw+s0f5kYmT60Zato+GtQ/55r+dN/wCEa1H/AJ5L/wB9VXtIdxfWKX8yMjJFIWPrWx/wjeo94V/76oPhvUP+eI/76o9rDuL6xS/mRjbj60m4+tbX/CN3x/5Y/rSHw3f/APPL9aPaw7h9ZpfzIxwxp4Nag8OX4/5ZfrQfD98P+WP60/aw7lKvS/mRl5I6UuTWn/YN9/zw/Wk/sG+/54n86PaQ7g69L+ZGbk0ZNaX9g33/ADwP50f2Dff88D+dL2kO4vbUv5l95l0ZrS/sK/8A+eB/OkbRL9f+WBp+1h3H7al/MvvM4001o/2Nf/8APA0v9i33/PA1XtIdw9pT/mRlleOlR7a2P7Fvf+eBpDoV6f8AlgaXPDuL2lP+ZGPtpCvFbP8AYd2OsBpDol0f+WDUe0h3K9rT/mRilaaUrZOi3Q/5YNTTo9z/AM8Go549yvaU+6MbbRsrYOj3OP8AUmozpdwP+WLUe0j3Dmp90ZeykKVqf2bP/wA8m/Kk/syf/nmaPaR7hzU+6Mry6XZitI6fMP8AlkfyppsZT/yzP5UuePcfNT7oz9tNZK0DYyj/AJZH8qQ2Mv8Azzb8qfPHuPmp90Z+yjy6vmxl/wCebflTfsUv/PM/lRzruHNT7opeXRsq6bOUf8s2/Kk+yS/882/Kn7RCvDuU9lBSrn2SX/nm35UhtJf+ebflS9ou4c1PuU9tOAqz9kl/55n8qT7NJ/cP5U+dD5odyGlxUv2eT+4fyo+zyf3D+VPnQc8O571+yxxc+Jf923/9nr6H6mvnv9l2NkuvEm4EfJB/7PX0ItbxasefUtzaCSD90cHtXimraTdXkszWes6jBIWJG25fGc+ma9tOCpB6Vx+q6Xo2kOjKsr3EjgpH5vv1PtXBjIVJcsoSskbUJQV1I+f9XfxfpF0YptR1Vh1VlnkII/Otj4X6n4kn+JWmR3t5qclsxbzFmkkKEbT1B4r6TEELKu6JDx3UU9IIUYMkSKfUKBXXCMorV3MG03oSCuH+MP8AyS/VPov/AKEK7jNcP8YP+SXar/ur/wChCtCT5T3U0nNJnNL0qBiHmiiijQCzQKjBpwNIY4000E00mgY4UtMBoLYFAupNbHF1F/vj+de9wvi1i/3B/KvnK4v2tZY2Xkg55roV+K+rrgCGDAGOhrgxVCVZqx34Zpas900eaWLVpTEFIZQcc/N7V2+m3A2FyPwHrXzPo/xd1SHUIjLawupbB25BIr3Lw14ltNcs2ezfZMo3SW7n5h7j1FRCLppRZy4ipFTO5UrLH6g9jXlvxA+H3zPq2iRL6zQIP1Arupb/AMhUbjG0ZIasfXPFkFhpczFv35XESdcntU1rJanBieV03zHiG3B5BBHY9qdgGp5Y5J5XlkYF3JZvrURt5McV5rZ8dJq7G7RSYoFvN3NPEEvtSJ5kIBTttKLeWnfZ5valcTaGhaXFOEEvpS/Z5fSlcm6G4pVFO8mX0oME3YCgdwwKTbz1pRbz+gpfs8/oKV0JMMUY96XyLj0FH2e4/uii6C4mBShRThb3HoKPs9x6Ci4cwm30pcUot7jso/Ol+zXP9wfnRdC5hmKcEpfs11/cH50v2e5/uj86Vw5xu2jbT/s1z/dH50fZrr+6PzouHOM20bPan/Zrr+4KX7Ndf3BRzBzsYEHpRsp/2e7/ALg/Oj7Pd/3B+dFx8wzbRtFSfZ7nvHR9muP+ef60XQKRHtFG0VIba4/55/rTWguAP9X+tFw5rsYcAV1vgLwwup3x1S+UCztjlQ38bD+lYOg6Hd67rEVnGmEJzI/91a7Px9rK6HpUXhzQSFkKBZmXqq/4mmj2sHRVOPt57I5zx94lPiHWBZWjEWFqcADo7Dv9K5vylC9qWG1mVfu5PekkSZVPymqbucOJxDqzbTMrVTstJCDjjrXIy6tdS6ZHYSlWhjJK/LyM+9b3iG4ZbUxsMFjXKgZNexg4vlue5lql7O7FpDS4pCcV6B6g1qt6bYG9mCjpnmq8EL3EgSMbmJ6V3ei6Y0FqqeWN3UkCuHE11BWR5mNxXs4WjuT2enR28QCLjArQjjAHSpFtplXiOnCGftGa8OU77ny8qjk7tibfal2juKUw3GOIzTPKuf8Anmam5POgwB2pCB6Uvk3P/PKkMVyP+WP60XDnEwPSkIX+7Ttk/eI00rcdoT+dMOcQquOlW/Dcdu3iqw+1uEiEoJPqRyB+dVltp5Dyuz6mrEVmE5Y/MOQfSqjNRdzWlVUJKXY9b8VeMrDwTpC6jq24jO2KJTlpGPauS8L/AB90jXNZisL6zksjcOEjkZgV3E4APpXDeMLC98XWtpFd3Z3WQKxk85B9a57TvAEdtMsl1OZNpyFXjmvTjiKPLd7n0yzKhy3uemfEmGCPxR5iyh5pkBdAPuY4HPvXJYzVy4ie5kMkzs8hH3mOTVb7PKvRc15s5qUmz5zE11VqOURuwUuwUuyYf8sjTtsn/PM1Nzm5hmxfSl2D0pdsg6RmjEv/ADzb8qRXN5ieWDR5YoxL/wA82/KjEv8Azzb8qYcz7htFNZBTiJP+ebflSYk/55NRcOd9xuwUbRQVl/55P+VJiT/nm35U7i5n3AqKTApcSf8APNvypMP/AM82/KkHO+4uykKijMv/ADyajMn/ADyajUOaXcTYKNg9KXMn/PJqTMn/ADyancrnYbB6UbF9KMyf88mpNz/88moE5S7i7B6UhQelG5x1jaje3/PNqY+eXcbsFIYwaduP9xvyoLY/gb8qCueS6jPKFNKVJv8A9hqTcf7jflT1D2ku5HspNlPL8fcb8qbv/wBlvyo1D2j7ibBSFBS7/wDZb8qaX4+635U9R+0fcaYxSeX7Cnb/AGP5Uhk/2TT1D2ku43y19BR5Kf3RS+Z7Gk80f3TTuxqpLuMMIz90flR5I/ur+VOMg9DR5g9DRdh7SXcjMEfdR+VIYI/7i/lT/M9jSeZ6g07sr2ku4z7PF/zzX8qPs8f9xfyp/mD0NJ5o9DRdj9pLuN+zx/3F/Km+Sn91fyp/nCjzV9KNQ9pLuRmGP+4v5Unkx/3F/KnmRTSbx609Q9rPuM+zx/3F/Kmm3j/uL+VSFx600uPWnqHtJ9xn2eP+4v5U37PH/cX8ql3ijetO7BVJLqQG3j/uL+VJ5Ef9xfyqbetPVVLDdnbnnHpTuzSNSTdrnq3wCiWOfXCigZWHPH+9Xs+Oa5rwPYWFh4fhGlwokUiK29er8dSa6TPPNe7h42pI+soR5aaTILpX2Hy32nHGBXmviUNH4it5kuWknkG0hm+76AV6e0Ku+5sn0FcZ8QLzw1o9lHe61cR21xGQYApAeQg5xj0rCvRlJOzO2i+WaNTTNG1faH1TVpjx/q4m/rXQxJ5caoGYhRjLHJ/OvJ/+F+aIq/6lDgf89/8A61P8I/HG18VeLrfRItIeE3BYLN54YcDPTHtW9JRirR/EdWnNe9JHrFcP8YP+SXar/ur/AOhCu4BzXD/GH/kl2q/7q/8AoQrc5z5QFLjNNH3aUVmMMUU4UUXAN9PDcVHtzTwMCkA4nNMOadSGgpCDikY5FOppFAupkaoMSL9KoI3FX9V++v0rOXpQdlPWNy/p75vIv94V6rY3NxYTRXNpK0UycqymvKNMGb2L/fFepoPkX6V5OMk4yVj5vN5unOLR1MnxAury1W2ngC3SHmUNhW98etZc08t3MZLiVnPbJ6ViXabUVx1DCtSDmFT7Vw1Jt6nhV8VVqKzZMFGKXaKUdKK57nFe4m0UoWilFABilxQKdikK4gFLtNLSjpSEIFpwWgU4UCbAClC0UtIVw20uKWipDmExS7aKf2oDmEC+1LilopC5gxS4oFLQTcMUYpaKBXDFGKWikO+glKKKKCbi8UuBTadQO7EOKiYF2CRjczHAA706R9tdV4U0eC0spNe1n93DGC0Qbv71UUz0MFh3XnrsjRjNv4H8Lm4cA31wMKD1J9PwrzgyTXl1LdXLF5ZW3Mxq5r+tT+I9Wa5kyIk+WFOyrUMS+WvPWtHKysdeNxV/3NP4UAUr1rI1jVYdOgaSY/RR1NaV5drDCzu2FUZNeZeItUOo3xKH92vAFb4ak6kjHBUHVnZlTUdTm1K5LuSEzwvpVXFIoxTq+kpwUI2R9dTpqnHlQhNM+82Keau6PYNe6jGuPkBy1TVmoRuyatRU4uTN/wAL6PtAuJV5YcV3VpAFjGBzVWxs1jjVAMACtVECAAV8xXrOcrnxWKxDqTuATFPC0tKK5bnFcaRTSKeaaaEFxpGaay06mk81QXG4pMD0pxpMUBcQim7RT8UYoHdjNo9KQqDT8UmKAuM8sUuwelOpCcUwQm0UbRRup1A7jdo7UYp1GKBXEAFBApcUYoC7G7RQEHpS0ooHzMaUFM2CpaYaLj5hm0elLtHpSmimK4mwUhQelPxxSUBdjdg9KTyxT6KB3ZGYx6UmwVLSGmO5CYx6Unlj0qUikxTuFyIqB2ppX2qU800jFBV2R7fajA9KdijFO41IjKj0phQCpiOKYRmmO5HtFK6Dylp22lkGIlp3C5WMYNNMVS0VVwuQlKTYPSpSKZQPmGbBR5YqSjFFw5iPYPQUhQegp9IaY+YZ5Y9BTTGvoKkpCKdx8zIvLX0H5UhjX0H5VLikIouHMyIxqew/KmmJf7o/KpcUmKdxczIzEvoKaYV7AVKRTadw5mReUvoKTy19BUtIRxTHdkJjX0pQtPIoAp3GpNHrHwV1aaQahpc8xZYlWWBD2GcNj8xXrPevl/Q9dufDmsQ6lZH54j8yno69wa9u8P8AxS8M67bgvqEVjcAfPBdNsIPsTwa9nCYiLhyy3R9RgMVGdPlm9UdkehxXlviT4GWHizV5NS1nxFqss7ngAptQeijbwK7k+LfDo667p/8A4EL/AI1Wk8f+EImKy+J9JVh1BvE/xrs5oy2Z6kaifws82P7Mvh09db1T84//AImtrwj8CdE8H+JbfW7PU7+4mt87UmKbTkY7Cut/4WF4O6/8JRpP/gWn+NOT4g+D5DhfE+lMf+vtP8afKaOUmtToga4T4z3Edv8AC/URK20yFEX3JatPUPiT4O0y3M1x4hsXGMhIZRIzfQLk18/fE34mT+O75Le0RrfSbZsxRt96Vv77f0FUScF/DSrzR2pB3qAH9KKaTRQBLilAozSikUJikpxppoAKaadTSaAMXVz+8X6VnrV/V+ZV+lUUoex2UvhL+lDOoQ4/vivVUHyL9K8t0j/kJQ/74r1Rfuj6V4+O+JHy+efFErXo/wBGP1FaFp/x7pn0qjff8ex+oq9Zn/R1+lcEtj5qZZpaSlrEgMUoFJSigBadTadSJAUoFJSjpQA6lpKWpEOpRSUooEwopaKQgFL3pBS0CHUtIKWkAUUUUCY4UtIKWgQtFFFIAooooADTGfHenHpSW1pLqN/FaW4JeRsfT3ppXNqVN1JKKNPw3ora7qoDKfssR3SP2+lSeNfEK6jcrpOmnbY2x2tt6OR/QVq+Ir+Dwnoq6Jpbf6XOv71h1APf8a4a3h281toke7VlHC0fZx36ksMIVaSYhF61LvVVJrnPEOuxWUZRCGlPQZ6UqcJTlZHj0qc6k7GR4u1V9wtYX4Iy+K448nmrNxO9xM0khySahIr6PDUlTifYYSgqUFoIBS0lPiQyyBVGSTiumUlFXOqUlFXYsULTSBUGSTXd+GtI+yxAyD5z1NQ6HoIiRZJEy59uldda24jUcV4WLxXMuVHzOPxnMuSJPHGFGKmFN7UoryGeAx2KXtSClzxSIuNNNJpxqNutCHcTNFFFUUFGKMUhpgBpM0tGKAEoxTgKXbQBHimkU8000wIyvpSgmnYoxQMKUGmkUUCHUmaXNJmgApVpKM0DFqM9afnioz1oAWkFApaYBRRRQAUUmaUGgApMUuaKAExSYpSaTNBQxqZT25puDVANNFB4pO1MaGmkpcZpKZYYocfulooc/uVoAiIFJTqQ9aYxCKbgU6imA3GaTbTjSUARmkpxpKYCYFIadTTTASkpaSgY1qbUhphpgMNJSmkNMQlIelLiimUMFKaWkNMCC4OIm+lVLJB5G7HJNWrniFvpVex/49R9a0Xwm0diVwNh4HT0rzzVVU6hJlR19K9Fk+4a871YY1CT6124L42e3kyvORS2r/dH5Va09V84/KOnpVQVc00/6QfpXrn1UloawAHaloFLig5XuFFFKBmpEFFLiigCQdacKTHNLSGIaSlNJQDCmNT6jagFuY2q/wCvX6VSWrmqf69apjih7HdFe6jS0j/kIw/74r1Nfuj6V5Zo3/ISh/3xXqSnIrxsd8aPlM7+OJDe/wDHsfqKu2n/AB7p9KoXxxbfUitC1H7hPpXDL4T52S6lkUtJS1gZBSijFFMYtKKSlpCFpR0popc0gH04U0U4UEi0o6UlKOlIBaKMUuKRIlLSUopAOAwKWjtQKBMKXFLikpCYopaSloELRQKUDmgBKCcClIxTGPBo3GkMd+MDr2rs9Lhh8HeHpNYvgDfTriCI9vSsvwro8dzM2q6idljZjecjhiKyfEGtT+JtWMrfJbR/LBH/AHR6/WtUkj6DCwjg6Xt579DPaW4v72S8vJGkmlbczNVjARDTAPK4NZOsautnbsxcbsYAzzQoym7I8p82Iq83cq6/ra2MZjjI80jp6VwM87XEpeQkknPNS315LezmSU5J6VVxX0GGw6hHU+rweEjSjd7iCmsKUmnxwvMwCKSTXZJqKuztlJRV2QoGeQKoJJOBiu28P+HfJAmnG5m7EdKPD3hnaUmuV+bOQpFdrDAIwABgCvDxeKu+WJ83mGNu+WDEgt1jQDFWAMUuKMV5LdzwG29wpaKXFIhhRQaMUiRDTSKeRRigfQjxRipMCmsMUxjaaetOoxTGNopcUYoAUdKKAKXFAyM9aa1OPWkIpgNFJRS4pgJSmiigYlJS0lAgopcUYoASm0/FNxQMSiiimAUhpM0vWgQUooxRTGgNIaWkJpDYnemmnGmmmMSiikJxTARqZ2pWPNJQAlJS4oxTKGk0EfulNKVpWGIhQBFSGnEYppqhjTS4oNLTGhhpKc1JigBhFNNPppFMQ2g9KXFIaZSG0GiigYhphqQimkUAMppp1IRTENpKdgUmKYDaKWkplFa7H7lqhsR/oq1Yux+5b6VWsObUfWtF8JrHSJM/3TXnusf8hGT616Iy8V57rS7dSlB9a7cF8Z7uSa1JGaOpq1pn/HwarVb08YuPwr2GfVT2NgdKUU0U4VJxhSg4ooAzQAmaKDwaKdwJh1p2aYDzT+1SNCGmmnGmHNAwNMNOz60w0C6mNqn+vWqYNXNVGZl+lUkGKDug/dNTRRnVIQOu8V6iowK8y8Oj/idQf71en142O+NHymeP95Eq33+oH+8K0rTmFfpWbff6j/gQrRtP9Sv0rglsfOSZZpaSlrIm4tFFAoJAU6kp9ITEFLiiigBadSYpQKRItKKTFKKkBaWgU7tQIbRSmnAcUiQ7UClxQaAF7UUgpaBB3paBS0hCjpQKKQ0ihWNT6TpM+uamlpbnaucyP/dXvVT55JViiUtI5wqjua6m/uB4L8N/Y4GH9q3y/M4/5ZitYRPUwOGjN+0qfCin4v1eGJI/D+kHFrAMTMp++3pXPQqIxmoo1AG48k8knqaC5uZjb2hDy46A9Kvlvoh1XUxlblhsUNc1iOxiByGc9BmvPL++lvbhpJCTk8AnpXd+JfCF5NZfbbc+dLEv7yMenqK87cEEhhgg8g16+Doxirvc+gwuW/Vopy3G5pDQFZ2woroNC8NzXsweYYjHP1rtqV4U43N6+IhRjdsyrDSri/lCxISCeuK7rRfDMdpGDKm5upJrcstNgs4gsaBfwq6AB0rwa+MlU0R8viswnV0WxDFEIwABipgPWijFcF2zyW23qFFKKKACnU3NLmkIDQDRmigkDRQaKCugU1qXFJTGNop1GKAG0U403FMYoooFKelIREetJTqQ1QxpFJTjTSDQAUUmDRTADSUveigQUUuM0YpDEppp9RtTGFFIKWgBKB1paKYhaSlFNNAwzTSaU0lMBKKKKChKRu1Opj9qYDDRS0UwEooNFA0xKVv9UtApW/1YoGQmmmnN1pKoY2loPSkpghDSUppDQMbTTzTqbTEFNNOpKY0Mop1BoGNNNNOPWkoAjNIacaaaYCGkNOpppgJSUtGaYEF3/wAe7fSq1gP9HH1qzdf8ez/Sq+n/APHqPrWi+E3jrEsmvPde/wCQpL9a9CIrz7X/APkKy/WuzBfxD3clVqsjJq5p/wDx8D6VV7GrWncXH4V7J9VN3RsU5aaDS5qWcgtApc8Ug5oENNFP20UgHgc06mjrS0ABpKKKBsaRTCO1SGo3oEZGpf64VTxVvUDmYVVHSmd8PhNbw6P+JxD9a9M7V5p4e/5C8P1r0scqK8XHfGj5PO/4kStfH9yB/tCtO2XEK/Ssu9HyL/vCta3/ANQn0rglsj52RLS0lLWRAtJmlpKBMUVIKYKeOlIkKUCkpRSAdSim0opCHUAZopRSEOC0tFJSAcBmlxSLS0Ei0h5opaQmJS4opaAAUtIKWgAzTWYCkZ9gyRmul0DQ9P19heElIoSPNQHgY7fjVxjc9XA4CeK1Q7Q7W20DS5PEmsJkqMW0RHLH1rjr/UrnVdRmvbrLSSNkDso7CvTdVRNRKo8S+RFxFGegHrWWdLgGcQqPoK6owPtFk/PSVN6I4u3spblfnDRqfWufazv9L8ZLJpxbLY2g87/avTZLJVOAvFYfiTZZ2S3dpc/Z7+HIhYDJOe1dNFQjudtPLaGDpOUSe115VS4g1GAWd5bHEsbnqD0Neb65p8Wu6u02mQCNT94jgE+tW9P0bULu7N1qM0jmTliTyfrXUW1jFAgCIBipqYn2b90+cx2dRUeSG5zWi+EVtf3lyQ7ZyBjpXUw26RLhFAHsKmCY6UoFedUrSm9T5OriZ1H7zExRtpwpetYnLcbikxT8UYoBDBQadRQO43FFOxRigLjc0tLiikSIaKWimMSkNOpDQAlJmlIpMUDuHWjFLilxTC42ilxSGkK40im05ulNqkUJikxTqSgBMUYpaKYDCKMUpooAKWikoASkK06kNMYzbikp+M00igYgooFLTEJnFJS02gAzSUUUxiUUd6KChpOKRuacRTcUwG4oxTqSgQ2jFLRTGhKGP7sUmeaVv9WKCiJqaacRTaoYnakpTSUAHWkIpaSgY3FIRinUhpgMopaSmCEoNLSGgGNPWkpxpMUxjCKaeKee9MNMBKQilopgxpGKTFONNpgQ3P8Ax7v9Krad/wAeo+pqzc/6h/pVbTR/og+prRfCdEPhLRrz7Xx/xNZPrXoRrz7X/wDkKyfWuzBfxD3Ml/iv0MqrOnnNzVarWnj/AEivZPqpbGsKUUgpRUnGLmlU0lCimIf1ooAoosBIBRQTiipGIaSlPSm4oGwpjDg0+mMeDQBjagP31Vh0qxfHM2KgFI74fCbHh7/kLw/WvSB92vOfDiFtWiI7c16On3RXj4740fJ53/EiVb//AFaf7wrUt/8AUp9Ky74ZRB/tCtWAYhUe1cEtkfOy2JRS0YpayMgpKXFGKQCin0wU7NAC0opKUUiR1LSUoFIQtOHSm4pwpCY4UtJS0iRRRQKWgAooopCFpabmnUAFWLGCK4u1jnkESHqx7VARimN1wOp6UtzWk0po6W1s9Je7S0gJubmQ4VRz+NXbnWrHSNSTQNOxvzuuXQcbv7tJpNg+g6esgQf2reqRHkf6lP7xrPkTTtCYiBftWpTnnJyzN6mu2lHQ/SsFBcqk48p0Cyb1zTHYBeazzem18qO8ZVnk42j1p99ci1tvMmOPat3JWPcdSEYt32G3l1HDC8j4wo/OuMEbXl493d8sxyqHooq1NLJdzFpCdueBmnAAVxVavRHwOdZu6rdKk9AAA6dKdSYpcVzXufH6vcKSloApCEFLRtooGBNIKWkIpgFFFFABSikxSjigQUUZpM0ABoFFAoAKKMUYoAMUmKdikxSAKKKKACm06kxTAaelMNPPSmGmUhKKKKYwooooASkp1JTAaaQ04ikxQAlFGKSgYtFAFLigCNhTakNMIqgAUUAUoWgYyinFaTFADTQOlKaO1A0IaSlNJTGIaaafTDTAaaQUppKYAaD9wUUp+4KBjDTGp5FIVzTGMpKUikoGIaSlNJimAlJS0GmMYetJSkc0lMQhpKU0mKAAU006m0ykIaaRTzTTxTEMNJSmkzQMU0w07NIaYEFz/qG+lQab/wAeY+pqe5/1LfSoNN/49B/vGtF8JtD4S0RXn/iAf8TST616Ea4DxCMaq9dmC/iHv5L/ABJehjEVYshi4GfSoiKnteJ1+le30PqpfCagpaaOlLUnEP7UqHimZoU0hEuaKbnFFUBIaBQaKgoD0pKWkNAhjHmmMeKeetRv0oAx7v8A4+W+lRqKkuebls01RzUy2O+HwHQeF1/4mQ9lrvU+6K4XwuM6j/wGu6X7orxsY/fPkc70rRK17/B/vCtWL7grJvf4P94VrRfcWuGeyPAZNRRRWZkxaKKKQCilxSUooELSigUoFIQop4GKaKdSJFooHNOxSAKWkpRSAUUtAoxQTbUWkp2KTFILCUuaXFJQIUtW1oFnDDHLrWoj/RrTlFP/AC0f0qhpOmtqmoLFnbEvzSv/AHVHWpNf1QandC0ssR6da/LGi/xkdWNVHc9TCQhTj7ep8i7ceJ5r9GmiUrcTcEkfdHYCrljp1roUJ1HUn8y7ZchW5IzXNRAxlWTgqciprm4nvJBJcyNI3qTVqrbQ9SlnXLFupq+hMGm1jXIpG+8zgKP7oq34qJGsLaqcrEgyfWn+Hbi2srma7uT80UeY19TWZdXL3t7LdS/ekbP0FDqaGlTGtYLnctZDAPloFApQK52fJN3YUU4CjApANpRRigdaBOwtFFFIQGm0rdKbTELSZoopgLSUZpM0AFJnmloxQAtFKBml20hgKDSUUxBRRSUDCiiigAooooEMNNYcU8000ykR0tO2imnrTGFFAp2BQA2g0hopgFIaWkNACU08UtFAAvSnU0UtACEU0in02gBAKWiimUIabinGmk0ABFNpaQ0wENJQaQmmAtMNGTSUxiUtFLigBhpx/wBWPrRQf9WKBjDxSGlpD1plDT3php56Uw0xiUUtJTAbQaKSmMKYetPph60CENFBopgIabTqCKY0NNNanGmGgY2kpaSqABQaDSUAQ3IzC30qvpn/AB6/8CNWpx+4b6VV0z/j0/4Ea0Xws2p6RZcPSuE8SLjVW+ld2elcR4o/5Cg/3BXXgv4h7+SfxZGGRzUtvxcL9KjqWDmZa97ofVT+E0h0paatOrI4gpU60lOjHzUAB60U5gKKdxDyaAaUjmkxUjFzSGikNADT1qNuaexptAGPc8XTU1etLd/8fTUi0PY9Cn8B03hUf6e3+7XbL92uK8KDN8/+5Xar92vDxn8Q+Qzz+Ol5FW8/5Z/7wrWi+4Kybz70f+8K1ovuCuKeyPAZLTqQUAc1mZMdTgKTGKWpEFLikp1AMBSigUtIQ6iilpCAU8dKaKeBQITFKKWlAqbiEpaWjFAgooooHuZV9ra2Md2zxEtbsoC5++D3/nVqK+Wa6khRchEVy3rnoKU+EdU13XLeW0s3ktGQiVzwuR05/GoNH0m+0aOaDV4JILrfsAdfvKo2g/Tivdq4fBrBKrF+/ZaX76fhyyf/AG8jvlhWqXPb+v6/M6B737LoosrA/vbjm4mIwQP7oqhHGEUAU4UteFc5alaU9GKKd2pBTgakweoAYpMGnUopC12GgUopaAKBCilzSdKKQBQBRSjrSEGKSnUlAhrU2nt0ptMBKKKKAENGKXFKooAQClC0uKcOlACYxRSmkpDEwaTFPFI3SmIbRRRQMKKKKYXEpaKKQhhFIeKdSEZplIbSU7FMIpgLRmkWg0xjTRQaKAEpDSmkpgJRS0lACilpB1paACmkU6igBnSjNKabTGITxTaWkNMAppNOpppjG0UuKUCgdxmPmpdtOxR2oBDcUUppKBjD1pT/AKsU18NkHoeKyD9qFyNM8w+VjzPOz82z+79ff0rqoYf21/etbXXt1+7saQjzdTVpD1pQNqgDoKQ1zitYaaaacaaaYhKSlpKYxKQ0ppKYxDTD1p5pp60CGmig0UwEpCaU02gLgaYafTTTKGUlONNNMQGkpaKBkU/+pb6VV0v/AI9f+BGrVx/qG+lVdM/49f8AgRrRfCbQ+EunpXE+KeNUH+4K7c9K4jxZ/wAhUf8AXMV2YL+Ie9kr/eSMAnBqS1b9+KgY5apLMf6SK9xPQ+qnsa4p1IKdUHGFPSmU5OtAD8ZooopCHnrSUd6KBgaaTilNNY0ARseaTNKRmkoAyLof6U1IlOuv+PpqRaHsehT+FHUeEh/p0n/XP+tdkoyK47whzeSH/Y/rXZL92vDxn8Q+Qzv/AHhehUvP9ZEP9qteL7grIuzmWMf7Va8X3RXFPZHz7JelKKbmnA1kZjic0optOHSgQU4U2lFIQ4UtIKWkIdS0lLSAcBWWNXunknMGn+ZDBI0bP5wBOOuARWmWCjJOK4+X7OpvHaSQTfancR7yFkXPTHvXq5dSoVOb21ulr36vya/4B14bDyrX5Y3Owtp0ubeOaP7kihhkdqlNVrCeG4tY2t+E2jC4xt9qsmvLqJKbSVjkmnFtNWClpKKggdTJM+WwHXHFScUYzSHGVpXPYfC/2dvDdl9nwV8oA49e9cv8UEg+z2LpgXXnYGOpXHNcpp+r6hpqFbO6eNT/AAjpVe7uJ7+6+0XkzzSdix6fSup14uHLY+lrZnRlh/Zpa2IgDilwaUcU7iuS58tuNFOAopaQwpRRRQIKdim06kMQ0UUUiRcUoFIKcKADFHaijtQA080m2looAaVo20tLTAaBinClxQKACkpaKQCGkpTSUAGabkmnGm0wCiiigQUUUUAFFFFACYpCKdSUxjKQ04imkUwEpCKdSGmMbQeaUjmkxQMQ008U40hpiYlJQOtOxQCE6UUpFIBzQMDQDmlNNAwaAA02nGmmmA2ikNBpjENJig0UAGKWiloGIabmnHpTaY0JQaMig0DIXJ3pGgy8jBVHqTXc2PwyhKx3d9cu1yYtpWPhR3rhnZ4biG4iGXhcOB64r1Gx8daRJYrJPN5cmPmjYc5rjxdSvTS9j13sfV5PSw0qbc7OXmefa7pT6Lqj2jncMblb1FZprX8SawNa1hrhFxEo2oD6Vl5QjBGPpXVScnBc+54uOjTjWlGlsRE02pGQdVOfaoyCOtanAIaKDSCmMKSig0xiGmGnGmmmISkoNJQAtNpaSmAUhpaQ0FDO9IRSnrTTTAKQ0UGgCObmFvpVbTP+Pdh6Oaszf6lvpVbTP9Q/++a1XwmsfhLp+7XEeLP+Qqv/AFzFdx/DXE+LR/xNV/65iurBfxD3sk/iy9Dm2+9U1n/x8ionHNSWn/HwK9tbH1U9jXBzTqYtPFByBTk602nJ1oAfRRRQA7vRS96SpAQ1G3WpCaYRmgBpph6U8imHpQBkz/8AHy1IOKWf/j5akHNSz0IfAdV4QP8ApUn+5/Wux7Vx3hIYuJT/ALNdeT8teLi/4h8bnX+8FW4/4+ov94Vsx/cFYsvN1F/vVtRj92K4p7I8NkgpRSClrIyY8UtIKWgTClFJQKQDh1pwpopwNIQ6l70lLjmkIoaszrYymLO7acYrkEk0w6LL9pe4Gp7/AJAB8mPeu9kjEikMOKzJfDtlNIXKYJ64rpo1ow3Prcizqhl9OUKsdyh4QeVg4fO3tmupNVbOyis49sS4FWs1hUlzSuj53H4iOJxEqsVZMKKM0tZnAFPFN20jzRxEebIiZ6bmxRZt2QElOFMVg6hlIIPQinipYgqG7YpZTMpwRGxB/CpjWdrN6lpYS7/4lI/OrpfGjajSlWqKnDdmFYTGWO2S0W+W/dQ++Zj5bDPzHk8j6V14NeftqN5HaWcphkjhgI8mfb+GM9812Wl6gL+3Eg6969bNKsa0lKKtv679+1tl01PXzXK8VguWVWNky+KKKK8U8AKdTadSEFFFLmkMQdaeKaBThQAuKKOlFACGkxTqMUANpKdikxQAUUUtAhKKXFGKAEopaMUDGUU7FJigBKQjmnYpKAG4opx5o20wG0YpwGKMUCG0UUhoAKbTqQimA00hpTSUyhKWjFFMY0ikIp5ppoEyEgg0qt2NPJ4pppgKaKQUtABSGlPSmk0FDTSZpTzTaYCnpUZNSUwimA3NKKAKXFMYtFGKKQBTDTqaaB6iUUUUxjSM01kGOlOJ6ig/dFBak1sR4xTT1p5pjHFUG43vTw4xhxketMzSZpgOZBtypyP5VHTgxVsilZQw3Lx6imMioNGaQnNMBDTW60/FMPWgQ00UGjNUAZxSHHakJ5ozQUJSMeKU01qYxho7UGjtQIQ0lKaSmAyb/Ut9Kq6X/qH/AN81am/1LfSqul/6mT/fNWvhZtD4TQ/hrifFo/4mif8AXMV238NcX4tXGpp/1zFdWC/iHuZL/Gl6HNOOaktP+PkfSmOetPtf+PgV7p9bU2NYU8VGKeKg4h1KOtGOKB1pgOopaKLgO70hoopANNFOpOKAI261G3SpH61Ee9MDKn/4+WpB1ps5zcN9aFNTLY74/Adb4S5mm/3RXXfw1yXhD7830FdcvSvDxf8AEPjM4f8AtJTlH+kx/wC8K24/9WKx5v8Aj6i+tbSfcFcc9jxGxRSiloHWsjMcOtLRRSEFAopaBCilFAp1IQtOFNpwpALSikpw6UhMUUuKSs/V7yW0jtxDJHEZphGXkGQowT/StaNGVeoqcd2OMXJ2RonjpSisvT76WW+a2kmhuVEe/wA2HoOehrUp16MqE+SQpRcXZjqwNYG7Xos2QvAtsx8s445681uOQibvSuWubq8v9cKaTbvNcRoVOwZ+XPpXTl9T2VVztfRr715NM9TK8BWx1VxorZGxoCbraS4jVYoZ23RwqchOx+hz2rYHSuO0DVzFeS20ylXeQl19G6HjtXSahfGxtVkWPzGZ1RVzjJNTioyxGKfIvi2/r9eu5jjsFWwuIdKotS8aytcsDfWLIvXHFS2+oSm6S3vbYwPICUIYMDjrWjXJKE6Elzf5/ijmo1Z4WrGrHdanns0euXNjHpkzyNaRNuSM9Aa63Q7E2VkqP94jmtTA9B+VB4qqleVRWZ62aZ5XzKKhUSSXYpX+oPa3MEEFuZ5JgxADAYA6/wA6fZ3xuZJIpYmhmjwWQnPB96y9ajjm1ay86eWBFSTLxSFCDxjkVJ4eeKNZ4SxlnV/mnZixlHY5P8q9CVDDrBKa+O1+t/ia72tZbWvqjy5Yd+xVTl/q5t06gYNLXjHFYSilopALSikpRQAtAoooAWikNIetAC0nagUZFACUopKUUCFooooASlopKBiUlLRQISkNOpDQMSloooAQnmkzQ3WkoAKKKKYhMUGlpO9AxGFNxTzRTAZikp9IRTGhpppp5pDTKIyKQipKbQIbilpaYTQANxUeTmn9aMUxjaaetPpKYCU0innpSUDG0UpFNNMBRQaKDQAlNpTTaZQUUgpaAGHqaVvuikPWlP3RTKGGo2qQ0w0xjKSnHpTTTGIaQMVOR1pTTKYDnXChx0P6VHzmpVx0PQ0xhtbB6imFhOaaaWkoAaaKWimIYetJTzSUyhpprU6mGmMaaKDRQIQ0hpTSUwI5f9U30qtpf+pk/wB81Zl/1bfSqul/6uX/AHzWi+Fm1P4TQFcZ4wONSix/zz/rXZiuK8Yn/iZR/wDXP+tdOC/iHuZL/Gfoc2/WpLb/AF61G/3qktf+Pha9xH1k9mawp600dKcKVjjJP4aQdaShfvUWAkFFFFIBcUtFFA0Bppp1NNADG61E4+U1I3WmP0NMRiTD9+31pUFEx/0hvrSoc1Mtjvj8COt8InEkw9hXXL0rkPCX+um+grr1+7XhYv8AinxWbv8A2krzjN3D9a2UHyiseX/j7hrZX7o+lcc9keLIdS5pKBWZA8HNFIKWkKwU6minUCFFLmkFVbq+FrdwRPgJIHLOTjbtGaunTlVlywV3r+CuCi27IuinjiqVndSXW6QR7IP+WZbq/vjsKuA5qalOVOXLLcUk1oxacOlJS1mQKTisDxDe26varKFcRTCR1PORgj+tbrjK8Vx1+RY+JEur+1N3ary0ROA4x0rrwknCrzJ2f9I+hyHAUsbieSq7L8y/oWp263k0MAVLeQ70XoVPcfTvXTryK89tR9t8QmaytTbQM+5Ywc7B6V6DF/qx9KMZJyqczd3/AF+IZ/gKOCxPJSldP8AlXdGV9a4fUvtmka1LJpk0yTsuSYQSQDXdnpXN363R1u8azu1gdYEIVkBDnnjJ6VvlsITqSVR2SV9XZbpa2T79jnyfH18DVbo9UZvhzTJZ7o3s7FvmOSeu7vmtzxIyGwiWQsB56E7TggZqfRWt204PA7O0jF5C/wB7f3zWJ4kk/wCJhB9p3m2BG/b6d6rmdLG80fsvS/kd9J1c5zJKs7FrR7uzi1qSMSSTB1zDLKxYp6pz+dbOsXklppjzQuqPlQGYZAycZrz/AFW7sU1qL/hHPNjhbA/0jrn8O1b93qsj6ebK+MDOFSQPG3ykZ6EHoa7pYWdarTxFVpx0v6J2btt6/fYrNsnWGxcadFuV7GvaX9wt/DCbyG+WXO4xpgx4Gc8dq15G2oT6VylhrFnHrO+0RUimXbKvTDDoR/KurDB1BHINeVjvZucXTWlvJa+aWi7ea16ngYzDVcNUUakbHF3Bt73xGYtXvHtLXaSZAM4Paq2lzNB4gMdnK00AfCORjcPWuo1PQLfUGDY2t3xS6boFvYNvHzN71PtocnKfZTz/AAbyv6soe9a3/BNZOQKkzWXcajdJqBtLKzWcqgdmabZjP4GrVhe/bI33RmKSJykiE5wR796wnhasIe0a09VfXa6vdfM+CcJJcxapaKK5TMKUUmaWgBaKSloEITSZzQ3SkFAxc0ZpKKYh2aAc02nCkMWikooAKKKKADFJ0pSeKaTmgQZozSUUDFzRmkooAQ8mjFV7q9FrNbxspJnk2AjtwT/SrGauUJRSbW+w7W1DFJjFOpDUiEpKWigBDRQaKYMSkIp1IaAGkUhpxpKZQ2m07vTaYCEUhWnUUxjdtGKcaaaAExTDxUlNYcUCG5ytJQOlLimUJmm0p60UwEoNGaWgaGmmGpKaRTKGZ5pc0UlADT1pT90Uh60rfdFMYw0006mkUxjSOKbinmkpjGEYptPamYpgJinSDcob8DSU5eVZfxoGQmkNL3pD3piG5ozSGiqGBNJmg0lABmmGnU1qYDTSUtIaACkoooAZL/q2+lVNK+7L/vmrcn+rb6VU0r7sv++a1Xws2h8JokYrivF4zqUf/XOu3YfLXE+LuNQT/rnXTgv4h7OSv9+/Q5pxzT7X/j4WmP0p9r/x8LXuI+vlszXFOFNFOFI5GOzQvBpKcOtAh9FIQTRSAeKXFApwFA0NIxTDUjCmEUAyNqikOFNTMDUMgypoEYcx/ft9afGeKjlP79vrT0HFJ7HfH4EdZ4ROZZPpXZL0rjPCA/eSV2a9K8LFfxGfFZx/vJWl/wCP2D8a2h90VjOM38P0NbAPSuOfQ8WQ8dKXFJSjrWZmA4pw5opRSEwxTsUlLSEITiua8TX1lBeWhvjvjQOWjB5OQMV0bgkcVwOvr9h8Xw32o2H2yzAH7pmwH9q9DATlTrc8XZ2f4q36ndgoRnU952N/w54hgu2Npu4TiJmOCV9D7iuoFeVWAbUfFD3NhZ/ZLZ3ysIOQg9M16jDnyl3dazxyTq8y6hjqUYSvEmpcUmc0oNcJ5wEVDPaQ3C4mQMPep6BRdo0jOUHeLsV4LG3ts+TGFJ9KtDgUvFQ3E6wRM7HAUZNLWTFJzqz1d2zm5tQQXd0JtQu45llKxRxglT6DGKo2qy6/qpt7+/h09ljyXl4BI7VXvNX3xXUaxlo3lMgcA/KexBqe10yPX4xcB9suMPjua+kxFWg6a5Vt/kttFpe9079D7/K8PHKpe2x0LRa0e9hNAu5bfV3tVkEkeSMjofelt4La+0+BPIujfTAlZGJCMc8nk9MVeg0yDQI5LmVtzYwKyW1SeHTreHy3WKEgxS4x+RpYPFQgp76uP4KW/dXauuqMq2EqZjXnicvh7vfY1dQ0fTbCS1Z0QhXzLu7jH+NYFxPbwaiqwqJrNX3hOmRnlTViTU4b/U4p9QWR7QY3ohwSKh+zWt94gP8AZkUiWZcbEk5IGO9ZQxFVUPZTfu9vne/r+h7WBy94Sj9erzvOK2/T1LN9dQarq0Umk6aLCJVCmNWJyfWu8sEZLSNX64qGz023t0GyJQfUCrwGDXj1qqm9D4XO82/tKopRjZIdSE4pwx3qOX7hx6VzI+fiuaVjKfSZr/XpZgtysYhUBonK5Oa0LSO3sl+zxq0bAkkOclj6k966PQZrMQL9rUyKFwQjYINYOvyRtrEAt+uScdwPevWrSqVaKTloktPQ/QMVw/QhgHU5neKTv0ZOOaXFNXgc07NeOfnwYFFFAoAKWkpe1ACHpTacaSgBKD0paQkUxCA0oNNLClBoAdmim5paQxwpD1oHWg0AH8NIKXtSUAGKMUtJQIaaBRg02QERnHpTGYuv3iW9xYykMyxT7m2rnA2kVcsNWttQB8ncpHZ1I/nWUXhOqoNQaQW+/wDeFPvY9qWc2n9tgaM0ptsjaZBhj619X/Z1Kpl8Zt+8lo76b3Ou0XBHR7s0VGp4GakBr5Q5eoUlLmkpiYlGKKM0DQUGkooEIaKU0lAxtNp9NxVDEooxRg0AFIRS0hoGJikNOpppiExRig0ZplIQjmmkU+mmgGMpaSlpgJSEU6mmgoaRTTTzTCM0wYw0p+6KCppWGIx9aYIjNIacaaaZQ00h6U40wmmAhNNzTjTcUxhTk++B68U3NKv+sTHrQURngmmmnOMO2PWmUxCYoxS0hOKYDTSUppM0xjT0pKU9KSmMSmmlzSGgQlFFFADZf9WfpVTSOWmH+2aty/6s/SqekH55v9+tI/CzaPwmq3C1w3i451FP9yu4fkVwvi7/AJCK/wC7XTgv4p7GSf7w/Q5tjk1Ja/69ahzzUtsP3617iPspbM2RTlpi8AU9aRwi0q9aKVRzQBIKKSigCQU4EVHmigB7EYpgbBpKMUDsK0gx90VDI/yngU5uKif7poEc/J/x8P8AWp06VC/+uf609TSex3w+E67wf/rJK7EGuN8G8vJXZdq8LFfxWfEZv/vTIG/5CEX0Na/YVjddSj+lbOOlck+h40hwp9MFPHWsiBaUUlFIQtOFNFOFIQhqvc2cN0uyeMOvuKs4pcUJtbFRk1qirbabbWp3RRBfoKvDpTKcDQ23uTObnuOFOFNFO7VJAtOFMFPFIQoqnqFsbm1eMdWGKuCloTs7mlOo6clNbo4J59UstOm0hI1+zzPlv3eWP0NbnhbTZLK3ZpBjfzit4xRscsgJ+lPVQOgxXRUxEpx5WfQZlxDXzCgqMopW/EyddsHvrMpH1ByPeuYu/wC2bqxh0ubLW8B/drt6fjXekA9aBCmc7Rmpp15U1ZDyviLE5dRdGKTXTyOd07w5GtgiXCAnqa1LLRba0k3xoAa0gtLjFZyqykePWzHFVeZSlo+gmKdiiisjgbA0mMilpQKQiAwEEmORkz/dNENmkcm8ku56s3WrFLtqud2sdssfip0vYym+XsFLRilxUHEJQKXFGKACgUtA4oEFJmlNJigQhNRsalxTGWmMjzThTSMUooAUdakHSoxTxSGLRRRQIKDRQaAEooNAoAKRhkYpaKARnXumR3B3YwfWmWemR2jFvvN61ovTa6PrFX2fs+bQtS0G04UYqpealBZTJFIsryOCwWNCxx+FTTpzqy5YK7EouTsi5RVe0vYb2NmhJypwysMMp9wanqJQlCXLJWYmmnZgaQ0tJSEAooooATNFFFAw7U2lpKZQUUUUwG0hp+Kaw4pjEpppaSgBDQKKKACmk0vWmmmVcSikophcWmmnU2gaEPSm040lMYh6Ujf6r8aXtSP/AKofWgREaSlNJVDQh6VGakPSmUyhKSlNJTAQ0qf61PrSdqWP72f7ozQMjc/OfrTKeaYetMGB6U006kNNANNNp2KTFMY09KaaeeaYeKYCUUUdqYCYooooEMk/1ZqlpP8ArJv96rsn+rNVNK/10/8AvVovhZvF+6zTPSuF8Wn/AImA/wB2u7rhPF//ACEV/wB2ujBfxT18m/3n5HMHrU1of9IWom61JaH/AEha9w+zlszaAyBT1WmLTwaRwjwKdTRnvTqAA0UGigB1FHekoGFLmm0UDGv1qJ/umpW6VE/3TQLqYDj9631oU0rj9431oUUnsejFe4jsfBg/1hrrz0rkfBg+SX6iuu7V4OJ1qs+FzXXFMrL/AMhOP6VudhWGn/IUT6Vtj7orln0PHmOFPGO1NFKKxMxaKUUooEIKWinUhXEFLSilpCClFJinYoJFp3akFKKQAKeKaKcKQgpQaSikA6lFJS0CCnjpTKeOlJiY4UHrSDrTqkOg2nCkpR1oELiiiigAp1Np1IBaWmjrTqACkpaKBAKKWlFIkbRSmigBtHWlpKZQwrTDUxpuM0wIx1qQUmKWkMUUtFFAhKDS0tA7DDTCcGpajdc9KYADkUtRg4NSBqAAjNJtp9JSERnisHU2f/hIrbybqO3byGyzgHuK2buQxwsw6isC20oeJL9opJLaNlUsGuCAPoK9DA1FSqc0trNbJ7q2z0PXyrAzxlRxi7WRb0Qt9qvvOlW4k3rmdPutxwPwrYrndKc205tk27UYrhOnWugQnFRjJqrWc15fgrei9FotjHMcLLC4h05C0UmcmlFch5wUUtFADaKU0lMpCUlFFMYUh60tIetABmkNFIaBiGkpTSUwEo7UtJTGNpDTj3phNABRSZoNMANITRTaChSaSg0lMYh6Ujf6ofWlNNb/AFY+tMBhpKWmt1pjQhPFNooNMYhNNpaKYCGjlYmP97gUqgswApJGBfA6LxQMi+tJTqaetUIQ0lOpDQAlNp1NplCUxqfTW60wG0hp1NNACGkpTSUwEf8A1Z+lU9J/10/+9Vt/un6VU0j/AF8/+9Wi+FmsFozVrhvF4/4mC/7td1jNcL4w/wCP5PpW+C/insZN/vPyOYdada8XC0rdKIB++X617q1PtZr3WbSmnVEpwKeKDzyQU8GoxT160WAcaKDRSAd3pKM80UDQUlLSUAxCOKhk+630qY1DJjY30piRhsNzE+9IOKC3zHFC8mpPTXwo7TwX/wAe8re9dXXLeDh/okn+9XU14GJ/is+DzX/e5FeMZ1RfYVuAcCsSH/kKj6VtCuWoeRPcdTgMU2nVkZCgUoFJSigQuKWkpRSEKOKM0UUEtjqdTKeOtIBaUUlLSAWlFNpRSBjqKSigQ4GnCmU5fekIWng02lBpMTHDrS5pAaDSF0FzSim06gBaWmZpy9KQC4paSlpAAozRSUEjs0UlKKBi0ZoooJFzSZoppNAwzSbqbnmkHvTBD80UnFKKQwooooAWlpKKAFopO1FAhaTFFFAyMrTAxBqU96iYc00BKGoLCogcUo5osFxkyeahU96wrjQ5PMJibiuhOMU3GauM3E68Li62Fnz0nZmTp2lNbtvkIzWr0FOprGhycnqTiMRPETdSo7tiZ5oBppOKUHNI5x4ooHSg0gEoopKYwxSUtJTGFIRmlooAbSUppKYxDSUppO9MBDRSUUDBhTCKeaQ0AMxQaWkNMBO1Np3am0FgaTNBpKYBjNIw+THvRmgnimBGaaaeaaaZRGTSGg0VQCUmadSqoHzP07D1oAAfLjJ/ibp7VDjmnsSzZNN70xiYplSHpTKAEpDSmkNMBppKUikpjDFMYU+mtTAZTTT6bQAlGKKKYDXHyGqmkD9/P9atyf6s/Squk/8AHxP9atfCzaHws1QOa4fxmu2+T6V3I61xHjPm8Q+1b4P+Ketkv+8fI5UmiBv3y/WmtRD/AK4fWveR9tL4WbKDNSqKjjHFTKKDg6i4py9abTl60CH4ooopCG96eOlN2804cUFIKQ0tIaBjHqtK3yN9KsPVaQfK30oEYp+9Sr1pD940qikejf3Udz4Q/wCQe/8AvV0w+7XNeER/xLG/366UcLXgYn+Kz4PNP96kQwf8hX8K2hWNbD/iaH6VsjpXLPc8moOHSnUg6UtZmItKKSlFSMUUUCloJAUtAoxQSxwp4plPFIApKWikIBThTRThSAWgdaKB1oAfiilzSZzSEFOHSm07tSAcKWkFLSICnU2lzQAGlHSkzSg0gHUtNzRvFADqKAc0UgFpRSZozQAtJRmkoJFNRtz0p9JQMaB7UYp1FMAoNLikoGJTqSnYpAJRS0lABSUtFACUhpaaT81AIWmNS7uKYeaYxtKpopKYh9GaTNIaAFJqNjTs80baY0MxmlC808CloKAUGijNIkSiikPWmAUlLmkpjENJTqb0oASkNLSGgYwnmjPrQRzR1qhoXrRikHFG6gYGik60vamAw0hpTRQA2mmn4ppFBSYyloIxRmmMaetZhae9lnMdybeOB9i4A+Ygck5q9cTCJNxPFYsgut04tUjkinO5lfI2n14616WBp87la1+l7W3130vY0p2ubEDOYVM23fjkr0NOqlYEQ2qQBt3ljBJq6oJrkqxUakktridr6DSKaBmnspHWk8zauFAHvWYgwE5b8qjZixyaCSetJTASijNBNAxD0phpSwpM5pgIaQ0ppDTAQ02lpKYxM0080HrSZpgLTDTs000AJRRRmmA2T7h+lVNK/wCPqb61ccfKap6YcXU31q18LNYfCzXPSuK8Y/6+Mn0rtK4/xiuTF+Nb4P8AinrZP/vPyOQbkUkQ/er9acy0RjEi/WvdR9vL4TWj+6KmWok+6KkHFM4GSfw0i9aC3GKQHmkIcTzRSHmimBLSUd6KQ0FFIaaTQDB6ry/cb6VOeVqCbiJ/oaAW5hfxH61IoqMdalWhnd0O58Ij/iVn/eNdF2rn/Cfy6UPdjXQfw14GI/is+GzP/epEdrzqZ+lbI6VjWf8AyEm+lbIrkqbnkVB1PxTBT6yMQpaTtSgcUihRSjpSCloJYo604U0UtBItPFNFOpAFFKKMUgAUooxS0hBRRRQAuacKbTlpCCnUmKdikAop1IKWkJiUYopaBCUCilHSkIKSnYpMUAPXpS0wHFOHSkAtFJSigVwopaKQgpKU0hpjEopaKBBQaWjFBQ2nUYpKBCmkoooGFFFJQAUxutONNNADOtKBzSYpwFMA200in0hoAZQfag9aBTAAKWloxQMSilxRQAlJS0UCEpD1paMUxjaKdgUmKAEpp606kNMpDT0pmeac1NpgIaBRRTGHamkUtJQMUUvakFL2oAaaaacaaaYBTadTaB2ENJilNJTGU9QQvAQvXrRo2uto8M6C3il85dp8xclfpVlxnOahMEfXYM11Ua7pKyKuU7RSZmkPAY9KvA46UgUA8Cg1nOXPK4DX+uaYelKaTNQAUhpaSmUNNIelBptMQhpBQetBpgBppNKaQ0DENITxQaQ9KYxtFFIaYBSUUUwEPSkBpTSYpoYN901S03m6n+tXW+4ap6X/AMfk9WtmaU9ma1cl4x+7F9TXW1yXjIfu4D7mtML/ABUerlH+8nJHFNT/AFq/WkJpsZ/ej619Aj7mWiNiP7oqQVHH90U+g897jqB1ptKOtAiSikNFICTHNFONNzQAhppzTutGKAGHhagn/wBU/wBDVhhVef8A1bfQ0AtzCHWpkqL+I1NGKT2PQ2R3Phcf8SlP941vg8VheGhjSY/941tjpXg4j+Kz4XM/96kNsudTY1tCsWx/5CDVtCuSpueRUHDpTqaOlOrMyFFOFNFLSAWlzSUtIQoNLTRSikKw9elOzSDpQKQhQaUGkoFAD80UgpaQgFLikFOBoAWlFJSikIKd2pKXNIVxQaWkFLikIO9LSUp6UCEpc02loAdmjNJRSActPHSmL3pwpAGKKKUUCAUZoppoAdmkJooPWgLgKWkFLQAUUtJQUFJS0lAgooooAKSlooGJSEUpNHUUCGbaXBpaWgY3pTTTz1plMQhGaTGKdRRcBKM0UUwDNFJRmmMWkNGaM0AJRRRQIKSlooGIaZUhpjdaCiNqbSk80VQCUUvWkoKEpaWimA2l7UHrR2oAaaSlpDTAQ02nGkNA0NpKWkpjGk9qUr8lNPWnn7tMCE000+mtTLIjSU40lUA2kNONJQAwikxT6RqYxhpKU8000wENNpxpMUwENIelLRQMjxSGnkU00wEptPpppgIaSlNJTARvuGqemcX0/wCFXH+4ap6b/wAf0/4Vcdma09ma9cr4yGbeD/eNdV2rl/GH/HpD/vmtML/FR6mU6YlHFMKjj4mGfWpmqIf60fWvoEfcy2NmM/KKfmoFbCinbuetBwEtA60zdShuaBEuaKjJop2As5ooFFSAlNLUp6VGaAHZzVe4/wBS/wBKnFQ3A/dt9KBoxFXNTKMVGAQx+tSLSex6D+E7vw3/AMgeI+5/nWz2rG8OcaPD9T/OtcmvArfxGfBZjriZC6fzfua2xWLpv/H9JW0K5am55cxacKZThWRkOFKKQUtACilpKWkSApRSClFICQUUZooJFpRTaUUgHUUZooExaUU2nCkIdSikpRSAWlxSUtSQKKdTRS0ALSZzS0lACGlpD1ooAdSmkpaQCjinDpSClBoC4uKKKU0gEpMU6koJG0U4immgApc02lFAD6KTijNBYUUU2gQ6koozQA0tSbjSnFNpjFpc0lITSAUHmn1GKdQApApmBTqbTATFFLSUCEooopgIaSlNJmmMUCjFApaAENJSmkzQAlN3GnZphpjF3GmsfWimNQMOtLSAUE0xhRikopgGTQaAKDQMQ0maU0lACUUUUwENNpSaTNACU2nGm0ykIRxmlzlaQ/do/hpjGd6axp3emN1pgMNJS0lMBOtIRzS0UyxMUhFLQaYEZFNxTzTaYDTTacaCPSmAyilpKBjD1pDSnrSGmAlJS02mAUYoooARhkEVT04Y1CcfSrhqnp//ACEZ60jszWnszXrmPGA/0KE/7Zrpu1cz4w/48YvZzWmF/io9PKf95RxclRAfvB9aleox94fWvoOp909maS/dFLgUxQdoxTgGoPPHZpQeaZg0oBzTAkzRTMGikIviigUUhjT0qPaalNNxQA3GKY43cVI1QyNt5pjW5lTYEzAetNWiU7pmPqaFqHsdz2O78PjGkw/Q/wA616y9B40mD6H+danavBrfxGfB4/8A3mQ7Tv8Aj9krZrG03/j7f61s1yz3PLqDhSjrSClHWsyB1FFKBmkSAp1JilxSEKKWkHFLSELTqbThzQAUopKUUiRaM0UlADh0pwpopwpAOoFFJnBpCHilpmadmkIdS0gpaQg7UUUUCENFGKMUAOozSUp4pAOFLTQ1G6gLElBpM0HmkDClHSkxSjpQSFNNONNJoGIOlLTN1ODZpjHGgUlFIYuaSiigQZxSdaDR0oAaaQHFHWimAopp60u7FJ15oAUGnAUwVKOlIBAKU9KKD0oAZSCjNJuxVAOxTaM0UANakpxptMoctOpo6UuaQhDTacaaetMEJTWp1MJyaZQhpMUtFMBccU006kIoAZRTsUmKBhRQaTNMYjdaSlNJTAbmg0YoNAIaaSl70maZYhppp55pjUIBD92l/gpM8YoP3aYDKY9OJxTTzVDGUUuKSmMQ0UUUDEoNGaQmgYxuaSnGm1QhtLQaQ8UwGt1pKU80UDGmmmlY00nimAlJilpM0wENJSk5pKYBVKwP/E0l/wB2rueKo2Jxqs3+7VR2ZtDZmwelc34v/wCPCP8A366M9K5vxf8A8eEf+/WmG/io9PKv95icY9RL94fWpGOaizhx9a+gR91LY01HyinjioUb5RTw1M85js0o61HS5NO4iQkDrRUfNFFwL4pabTh0qSkIaSnGmkc0Axj1BO2IzU7Cqtz/AKs0xx3MsnLGnrUP8RqZTUvVHc9jvdCH/Epg/wB2tQ9Ky9CP/Ergz/drUNfP1v4jPgMdriZvzHab/wAfTVs1j6V/x8Pn1rZrlqbnm1BRSjrSUDrUED6UUgpwpEhSikpwpCAUtJThSEHanLRQKBCmgUUUgFNJS0UCFFOFNFKKQh9IetA606kAg6UtFFAhwp1MFOFSSLRRRSAKKKKYBSk0lFIAFLSCloAfS0lFIBaUU2lFACmmEZp2aWgCHpSilZeaQUwH0UUUgCimmkJoAccUhNNFFMApKKKYAaB0paSkMBUgpopaAHUHpTaSkIQ0w08imd6oBQeKXtTaWmMDScUUUCClHWkooGKaaetLSGgaEqPvTiabVDDNFFFAC0hooNACd6KKQ0xgelNNKaQ0DEoopDTAKQ0tITQA2kpabTKCmtSt0ptMYUH7tFDdKAI2602lPWmiqKEzikNLSGmAlJSmigBtFFBpgMNJTu1MPWmMDSGg9KbTGFBopDQAw0lOPSmmmAlNNOptMBKKU0hpgJ2qlZf8hWX/AHavdqoWRxq0v0q47M2hszYrmvF//IPT/frpRXN+L/8AkHL/AL9Xhv4qPSyr/eonEMajH3x9ac9MT/WD619Aj7qWqNZE+QfSpAnFCf6tfpTxTOB7jfLFAjqSkNBI3y6KdRTsBNilpuaUVIxSaQ80ppKAGtVS6P7s1aY1Uuv9UaZcdzJ/iNSgVF/FUq1J2PY77Qx/xK4P92tSszQ/+QXB/u1pGvn638RnwON/3ifqS6V/rpPrWuDWPpX+tf61sCuWpuebPcdQOtFKOtZkDhSjigUUCFpwNNFOpEi0opopwpBYXNOplPFArBS0YopCClpKWgQUo60gpaAHA06mCnCpELRRRQA4UuaaKWkTYdRTcUUgsOopBWTNfXj6pPb2z28aQhT+9zk5rejQlWb5Xayu7/d+pcYOWxrUuKqadeNe2vmMoVgxU7TkEg9R7VcFZ1ISpzcJboiUXF2YAUYpQaQmsxDs0tRjJNSAUAFKKMUopCEpaKKAQhGaTaKdSUDuJijFOxRigBhWmEVKRTDQA0daKXFFMBtLS0hoAKTFBpRTAWl2mkp9SA3aaNpp1FAEZppqQ+tRmqEJmkzRSUykOzRmm9aXpQOwtFIKKAsLTWp1NbpQCGUVl69ctbWUTLM0IaZVZ1GSAc5qHTLsy3oS1upbu32nzHkXGxuwzgV6McDUlhvrHTXo+nna3or6mypNw5zZNJmlNNrgMh2aKSimMKSlNNpAgpCKWlFMZH0ozSnrSUwA02nGmGgApDSimnpTKGk5FJRRTAKG+7mkpW/1J+tMZEeaTGKWimNMZQaKDTGNNJSmimAlNNLSUDEzTDTjSUwGk02nEU2mMKSlpKBiGmGnmmGmISm06m0AFJSmkpgIRVGzH/E2l/3av1Ss/wDkMSf7tXHZmsNma1cz4wbFgo/2q6cjiuW8Y8Wa/Wrw/wDFR6OVu2KicUxpi/f/ABpSeaRfvD619Ej7zobMX+rX6VIOKji/1a/SpaDge4maM0lA60CHUUUUwH805aKWhjQGiiipEMPJqpecRmrRODVW8P7s0FR3MrualjGWFR9zUkZ+YVMtjtex3uiHGnxD0FaTVm6MP9AT6VonmvAqfGz4HGfx5epNpP3n+prYHSsjSfvP9a1x0rmqbnnT3HUDrSZpR1rMzHinCmg0tILDqWmg07tSIAUopKUCgY4c07pTV4pwqSRaO1LRQIbSiilBoEFLRRSAWlFJSg0AOFJQDS4pCBetOpo606kSxM0tJS4pBcHYIpJ6VxWr3FjJqV79qh3mVVEcg6oQOorrb4MbdgnXFcda3ljps94NV0wXzSqRHuYjYfWvTy+rOjJypys+/wA0/wBD67hrLaONqS9rK1uh0egaol9YRqVWOVBtZFGBx3FbANcP4SimW6Mm0+X0rtwK5MTb2ra6nkZ1hKeDxk6VJ3SHfSgDnmlAp2K5jxgApaBS0ihKUUYooEFFBpDQAtJRRQIWijNFA7CUw06mt1pgNoNLikNABQaBSk0AJQKKAaYC0tN3UuaQBmjJo60UAJTD1p+aYTTGIaKaTShqYxQKXFIDmlzQLUQ0hNBoPSmMTcaQ80U12wpx1poZzviHUgGiSMjdFKr4I647VW0zWD/bEhdfKWcfOmMZb+8Kj1a1uYrxbmJdxRtwyMg0ktxf+IdYjurqGONo1CHy02gge1erCp+55L6Wa+/X80fo2AyzLKmVOpUkuaz67P0OtR96gjvTjUVum2MA+lTV5b3PzqSs2kMzSjrSUo60gFNJSmkoBCUUGimMQikpTSUAIaaadSGmAw9aQ9KdjmmmmURmjNKRTKoBc0pP7g/Wm5p3/LE/WgaI6DRSE0yhD0php5PFMNMYCigUUANNIaU0lMBhpO1ONJTAaabTzTcUxoSkpaSgBp6U0049KaaYxKQ0tJQIQ0lKaSmNC1QtP+QxJ/u1fqha8axJ/u1ceprDqa46Vy/jPiyU+9dQOlcx4z5sVHvV4b+Kjvy3/eonCk80A8j60jdaQdR9a+jR950NyH/Vj6VJUUP+rH0qSkcD3CkzS0hoAfRSZooAnAoxQKWkAGkpTSUDRE/3qp33+rq43LVTvf8AV0FR3M6nx/6wUw5FOj++PrSZ2PY9A0b/AI8E+lXyeDVDR/8Ajwj+lXJGCqc/hXz0/jZ8Di3etL1Lmj/xH3rWFZ2lxeXECerc1oiuafxHnT3FxSjrSUo61BmPFLTaUGkAtPplOpEscKUU2lBpCH0q0lKKRI6iiigBDQBTsZFGKAsJSiiigQtFFKBmkADpTx0ptOqSQ706m04UCG06kxS4oAay5qrLpNrcNukiUn6VdxTgcChSa2NKdapSd4OxWt7OK2TbCgUewqyopc5pQMUm29yZTlN3k7sUUUUVJIU6m5pQaBC0neloxQIKQ0UGgYlFFGaACiiigBDSEU7FGKYDcUhHNOIpp60ANpKWkoAKSjNHXpTGFLmkwaMUCHZxSbqKaelIBC9M3UNTSasqwuaWmjmnUxig4NOBzTKMkVIDqKTJpaYhGFMIqTrSEUAkQmJWXDAGkWCNDlVA+gqXFGKq5SlK1kxtLS0UhDCaBzQeaMYpjsKaQUUlAxT0pKKWgBp603NOPWmmmAmaM0UhOKB2DvTCKUmkJplCEcVGRTye1NxTEMNLyYT9aUilx+4b6imOxFTSc0pNJTGkFNNLmkplAKKTNKOaYAaaacabmgBp6UnalIpmaAA9KSg0UxoaaSlNJTAaTTTTiKYaYwptLSUxBRiiigBKo23/ACGW/wB2r5qimItVVjxvGKuPU2gaw6VzHjH/AI9E+tdNmuY8Y82sf1rTD/xUehlq/wBqicLjrSAc/jUpHBqIfe/GvoUfedDbh/1Y+lSZqKH/AFa/SpaDg6iimtS0wk5oEPBopMUUCLIpaaKWkUBphPNPpMUARkVVvB+7q6elQXEfmKAKCofEjIxmnIMOMetdvZfC3Xr7T4by3SHyp13JmTnH0qeP4T+II23SpAFz2krF1obXO+UbK4mknbp6Z44rUh0q6vLL7cFYQB9q/L97HU16B4V+F9paWJl8QymdgBthi6fjXWDwvA3khbcRRQjEcUeQF+o7mvNVG8m2fL/2epVZSlszyeBSqqGRkOOjDFT16Tqfha3vLNoyhDgfIw6g9q82ulewupLW6XZLE21lNcdai4O54uNwboO/QWlFVGuyDwKT7W2OFFc/KzzrF7ilFZ4vH/u077dJ/dFLlYrF8U4Gs77dJ/dFH22T+6KXKxOJpUCs8X790/Wl+3v/AHP1o5WLlNIGnA1l/b3/ALlL9uk/u0uRi5DUyKMiswX0v9yl+2yf3aOVi5TUBFGazBfSD+Gl+3yf3aXKw5TSorN/tCT+7R/aEn92jlYuU1M0Csz+0JP7tKNRfutLkYcpp04dKzP7Sf8AuCl/tKT+7S5GLkNKnA1lf2k/92j+0n/uUcjD2ZqinVlDVH/uClGqN/cpcjJ5Hc1KKzP7Vf8A550n9qN/zzo5GPkNQdafmsj+1G/550/+1G/uD86TgxcjNWjNZf8Aajnog/Oj+1H/AOef60uRhyM1KKy/7Ub+5S/2o39z9afIxcrNUGkrL/tRv7n60v8Aap/550cjDlZp8UhNZf8Aajf3P1pDqjD+AUcjDkZq5FJkVlHVW/ufrSf2q39z9aORhyM180ZrJ/tc/wByj+1j/co9mx8jNbNGayf7WP8Acpf7WP8Azzo9mw5GahNNPJ4rMOqk/wAFA1Q/3KORi5WaNFZ39p/9M6adTP8Aco5GPkZo8UvFZn9pN/cpRqR7pT5GHIaVKapRX6SHB+U+9WPOXaSSMVPK0LlZJTTUBuR1BAB6biBn86HlePb5qMm77pI4P0NVyStexr9XqpXsPNNpDMoXLEVWe+QN8oJppMhRZaFGaonUgP8AlmaT+0f+mbU+Vj5GaFLWeNSH/PNqP7SH/PNqOVhyGiMYp3GKzf7SH/PNqT+1Fz/q2o5GHIaVIaof2qv/ADzakOqp/wA82o5GHKy/RVD+1E/uNSf2on9xqOVhyMvmkqgdVT+41H9qJ/cajlY+Vl2lNZ/9qL/cNH9qL/canysOVl6kNUv7UT+41J/aSH+A0cjDlZezSnmqH9oJ/daj+0U/utT5WHKy70pDVM6jH3DU06hEf735UcrDlZczSGqX29PQ/lR9vT3/ACp8rK5S5Taq/bk9/wAqb9uj9/yo5WHKWjRVX7bGfWj7bH2zT5WPlLJp3/Ls3+8Kpm9j75pwvoTbsMkHIo5WPlHnFNNQfa4/U/lTTdR+p/KnZjsyfNIag+1R+v6Ufao/U/lTswsybNGarm6j9T+VN+1J6n8qdmKzLRNI3FVvtSep/Kj7Snc0+VhZkpNJmovtEZ/ipDcR/wB6izGokxNJUP2iP1pDcR/3qdmVYmNJUX2iP+9TTOn96izFYlNMNM89P71IZ0/vU7APptMM6etJ56etOzBIkoqPzk/vU8SD1oK5R3aqN7GWAdfvKeK1LCyn1O+jtLRN8shwB6e9ejaT8Po7UiSaNZ5FA5c8Z+lb06bep6GFwkq2vQ8ytbgTwhujdCD2NYHjD/j1jr2XxJ8PBdMb3RjFBdkfvYCcLJ7j0NeReNdOu7NFgvreSCVTyrjr7j1rSnHlqo9HDYWVLExfQ4I96g/i/GrbRHPAP5VC0LbsAV7aaPr2mka8H+qX6VJUdvnylz1xUtM896Maab/FTmpvekSSUUgopgWMUYpRRSKQmKKDSCgAIphApxzSBaT2Kj8R9FeFjv8ABulsR1t1q1JKItQt0IBDZPPqP/11neBp1u/A9g0bbvLQo3sQTVy/ilE0M0Bw0ZO7/dPXFfOr+M7nrTV6Wh1enRLPETKw4Oa149rjOR8owQa5rS9QigtmnkdQqjJA5AHp/nqadrPia306ykvHOxIU3Mg5Pr/L+depzJnj8sr7EfjLxRb+GrHagWW+l4jjz0/2j7V4vd3lxqN9LeXj75pTljjFc7q3j6XUtUuLyZPM8xyV3N0HYVR/4TQf8+4/OuWrTqSeiPExuHxNd2UdDrKXNcj/AMJr/wBMB+dH/Caf9MB+dYfVqnY8/wDszE/ynXbqUGuQ/wCE0/6YfrSjxp/0wH50fVqnYX9mYn+U6/NGa5H/AITT/p3H50f8Jr/0w/Wj6tU7B/ZeJ/lOuzRmuSHjPP8AywH50v8AwmR/54j86Pq1XsL+y8V/KdbmlBxXIf8ACZH/AJ4j86VfGX/TEfnS+rVexf8AZeJ/lOw3Ubq5H/hMv+mI/OnjxgMf6r9aX1ap2D+ysT/KdXupd9coPF4/55frTh4sHdKPq1TsL+ycV/KdTupdwrmB4qUj7lOHihe8dH1ap2D+ycV/KdNmjdXOf8JNH/dNOXxLD3B/Kj6tU7C/snFfynQ7qUGue/4SWD0P5U4eJYO6sKX1ar2D+ysV/KbxNJurDPiS3PZvyo/4SO39G/Kn9Vq9hf2XiV9k3M0oNYg8RW3+1+VOPiK27Fvyo+rVewv7LxX8ps5pc1if8JDbf7X5Uh8RW3+1+VL6tV7C/svE9jczS7qwf+Ejtv8Aa/Kj/hI7b/a/Kj6rV7B/ZeK7G+Gpd1YI8R23+1+VH/CR23+1+VL6rV7D/svFfym9mjdisE+I7b1b8qT/AISO2/2vyo+rVewv7LxX8pv76MisH/hIrb/a/Kj/AISKDtuo+rVewlleK/lN4kUm4Vhf8JFB6Gl/4SKD0NP6tV7B/ZeK/lNrijisX/hIYPRvyo/4SCD0b8qPqtXsV/ZeJ/lNmgmsf+37b3/Kj+37b1b8qPq1XsSssxX8pr5p2ax/7etvVv8Avml/t239T+VH1ar2G8sxX8pr5ozWR/btv/eP5Uf27b+p/Kj6tV7C/szFfymxu4pM1lDWoD0b8xTl1eInlh+VH1ap2H/ZmJ/lNLNGapLqUTfxil/tCL+8KPq9TsL+zcT/AClzNdP4E0SPxDqk8d3LiG1VXMY6vk/yrjPt8R/iH5V13wvug3jyNEkA32suR/exjinHDyUveR04XLqsaqdSOh5d8TLu9m8f6hDPuijtZPJhiHARB0xXT/Bia613XZvD+pmS5094GcFjzEy4wQe1et+NvhHpHjPUBqTTPZ3e3a7IMh8dCR61e8EeA9K8C27R2BNxeT4Es7DqPT2Fev8Au3TtY+gdOLVmtDy7XrB9I1i409pPMELYDDuKzRV7xffIni3UhM4Di4YEe3asNtVt06yCvHeHnf3UfOVsurub5I6F/FFZh1y1BwZR+VH9t2n/AD2X8jS9hU7GSy3FfymmKXFZf9t2n/PYfkaX+27X/nqPyNHsKnYf9m4n+U0iaSs/+17U/wDLYfkaP7Vtj/y2H5Gj2FTsH9m4n+Uv004qn/adr/z2H5Gj+07b/nqv5Gj2FTsT/Z2K/lLnFLiqJ1S27Sr+VL/alv8A89V/Kn7Cp2D+z8T/ACl3bSbap/2rbf8APVfypP7Vtv8Anqv5Uexn2D+zsV/KXKSqf9qW/wDz1X8qP7Utv+eq/lR7Cp2D+zsV/KXMUVS/tS3/AOeq/lR/aduf+Wy/lR7Cp2JeX4pfZLtJVP8AtOD/AJ7LSf2nb/8APdKXsZ9hfUMT/KXCcUZqn/aFu3/LdKX7bb/8/CUeyn2D6hiP5S1mkJ96rfboD/y3Sj7ZAf8AlslP2U+xX1DEfyljdigkYquLiE9Jk/OkM8f/AD2T86PZT7C+o4j+Un3CjcKrfaIh1mT86DdQ/wDPZPzo9lPsJ4LEfyljNJUH2uH/AJ7J+dIbyAf8tl/Oj2U+w/qOI/lJyaKr/bYD/wAtV/OkN5B/z1X86PZT7E/VK6+yycmiqv2yD/nqv50hv4B/y0X86fs5dhfVK/8AKy0RScVUOowj/lov5006jb/89B+dHs5dhfU8R/Ky2XpC+aonUrb/AJ6r+dH9p2w/5ar+dHJLsP6rX/lZdzSVROq2v/PUUv8Aadoesop8kuwfVqq+yy5RzVI6paD/AJaik/ta0/56rRyy7B9Xq/ylxqbmqh1W1PSUUn9qW3/PVaOWXYX1at/KXM0h5qr/AGjbf89V/OkOpWw/5ar+dPkl2D6vV/lLWKOKqHUrU/8ALVab/aVt/wA9l/Ojll2F7Cr/ACsu0Z96pDUbb/nsv50v9o23/PZfzo5X2GqFT+VnpPws1TSrXVprfU2EdzPhYJX6e6+1e3xW6KRtAr5G/tG3DArMoI6HPSvafhV8RxqkX9i6lOJLmFcwyE8uvp9RXVTk7WPfwE+WPI1Y9PubBZlbkrnkY9a4XxZoK3qKtwkcwXOC65YY6Yr0BbhZEG09aw/EzxralsjCg7v8KqXc9iPxKx4xJodnJKf9DiXnoExXNeM9EtLTTIpobdEbfjIGK9AC8AgY3EnHoK5f4iEJoduh+80uR+VY06kvapJnv1rexv1PMlXHQUtOFBFe0fPMiamDrUrCmEUAPAooFFAFilFGKUUDQUYoooGNI5pyrminA8UAtzvPhx4rTRrltOvXItrhsg54Vq9eCLIgkjIdT0I718zjse9dPofjrWNEQRxzedD/AHJeQK8nE4RuXNA7qOISVpHtMtkrqy7cBiCwB+9g5Gaw/F6GPwjqzDkm2fcfwrmYvi9J5YE+mqW9UbGazvEnxMj1Pw9fWSWTRtcQlN27pXPTo1lNcyN3UpPY8ckkO0CoiTStkmkwfSvcSVjBuL2E3GnBsUzaaXafSnYd0Sb6N5FMwR2o59KLB7o8OTTt9RjNGDRYLxHl6UP71GFJpwGKXKVdDt1KHNMwfSkINHKK6Jg5pwc1CDTgaOUFKJMjtmpC59agV6d5gpWHzImEhp4kNVvMxTllHeixLaLQlY05ZDVcSjvThKvqKLE3RZD0vme9V/PX1pPPWiw7otBz60u81V+0Ad6PtA9aLCui35nFJ5nvVYXC96PtC+tFg0LBlI704SE1UM6H+IUouFHemkDsW99HmD0qp9oX1pftCf3qdhaFrzPSk8yq4nj/AL1L58frRYehN5hoEpqDz09RSeenrRYNCz5tHne1VvtEeetJ5yetFkGha86nLNiqonTPWn+dH/eoshWTLXmZo8z3qr50f96lEsf9+iw7IsNLQJah82L++KcJYR/FSsKyJhJ704S1B50PrQJov74pWCxY82lEozzVfzov74/Ok82L++PzosOyLizAHipVnFZ3noP4xQLpB/EKLDsjWFwKcLvHSsc3a/3hSfbF/vj86VhaG2Lr3rqfAV+bPxNHfLhjbxsQD7kAj8jXnq3ik43iuk8IanBb6jL58gVXjwGJ4zkVlUXu6F04RlJJn1PYa3ZalZpLDOmWGShPIqtqmt2elQmWR1dx0RTyTXlNrewqgaK4UjHVXp13fxNES8y/UtXl+1lexv8A2cua/NocD41umfxNdzFstM3mHHYntXLvMx71p+J72CbWJdkqsFAGQetYLTxg4DCvXpfAmYzSjJxRPvo8wd6reev96k81D/EPzqzO6LYk9KXzKpiVezUvnD+9+tK1x6FzzqUTVS81P7wpRMufvCnyi0L3nmmmc1V88eoo81f7wosLQsmc0ecarGVf7woMqY+8KLEss+dSGY1W8xP7wo8xP7wp2ET+fSefioPMT+8KQyLj7wosZlj7Qe1L9oPrVXzF9RS+YnqKLDsWPtNH2g1X8xPUUu9fUUWCxP55NKLjFV9y+ooLL6iiwyz9qNJ9qNV9y+opCV/vCnYC0Lwig3ZJ61UyvrRlfUUWKLRuSTTxMpHWqBdR3pjSj1pWHY0Gk9GqJ5iOjVRMo9aBMo60WCxa+0N60huD61WaZSeKiaUHoaViki20+aZ5/uaqGQY5NNLj1osOxbM+fWo/N9zVffmm7qLDsWTKKaZOOtVy4pPM96Vh2RKZCD1ppkz3qMsDSZFOwcqH+Zil83NQk0gbFFg5UTGQ+tMMhPemE0maLIXIhTIfWlE2KjpCKdkHIiXzz70jTH1NRUUWQvZpknmt6mk8w+p/OmjilNHKiZU0PWVvWtvwvqE1p4k0+aKRkZZ15B7Zwawa0NHkEer2jHoJlJ/OiUVZijCPMrn0zb+J7+OYhX3JgjnqKbeatd6jD5U3+rzkgd65+31WyR9z3EfX+9VuTxHpNuhZ7qMYHY5r56cql7HpqjRjqi4sDM2415r8QNUW81KO0iO6O3HJHrWtrnxBSSB4dKQgnjzG9PauAkdpZC7nczHJJrswdCSnzyOfFV1y8qICKaalNMNeueSMPSminGkFAh1FGaKALGaWkoJwKBoKKTNGaB3FoBpvWloAeGOKXcaZRSaAeWNRu2eDS0001oMga3j/ALi/lTTBH/cH5VYpNoNAXsVzbJ/dFJ9nj/uirR4qM0E8zKzwoB90VE0a/wB0VaYZqIrTDmZW8oZ6UeUPSrG2jFAOTIBCPSjyRU+KMUBzMi8ukMQqYCgigXMyDyh6fpSeX7VMaMUBdkXl+1HlVLRSHzMi8oU4RD0pxpc07C5mJsHpSFR6U/FNIo0HzMYEFO2D0pQKdTsg5mN2A9qPLHpTs0mTRZCuxNi+lGxfSjNKpyaLBzMTYPSjYPSn0hNFkF2N2D0o2D0pcmjNAczAIPSl2D0oBozQHMw2j0pCg9KdSUJBzMb5a56UeWPSnUVQ+ZjQg9KdsHpRS0g5mJsHpS7B6UtBzTFzMTYvpSeWvpS80CkF2J5a+lJsXPSnUuKAuxuxfQUbF9BTqKA5mM8tfSgRKO1PoNAczGlF9Kb5Yz0FPopBdjdg9BTkJTgcUUUWDmZMl3NH/q5nX6NSve3Drh5pGHuxqCilyo09pLuBAbrzTfKXsKdQarQjmYwoPSk8lPSn07FGgczI/KQUeWtSYpKA5mN8tfSjy1pc0vamHMxnlrS+WKWipDmY3yx/k0nlLT6XFCDmZH5a0eWtSYpMU7BzMZ5a+lBiT0p1FArsZ5SelHlJ6U+igdxnlJ6UeSvv+dPpaAuR+UPf86PKFSUUCuyPyR7/AJ0eSvv+dS0UBci8lff86PKX3/OpcUlKw7kZiX3/ADppiX3qU0mKA5mReSPf86b5I75/Op8UYoDmZB5Cjpn86DAp9fzqximnrTsHMyuYF9/zpv2dff8AOrFIaVg55dyAwDsSKT7OP7xqelxQVzyK/wBnX1NJ9nX1NWdtG2loHOyt9nHqaPsw9TVjFFMftGQG2Hqab9lHqas0uBSF7SRW+yg/xGk+x/7Rq2BTgtAe0l3Kn2MetL9iH941bxzTwKB+0kUxp4P8Zp39mD++avKtPpB7SRnf2Zj+M0h07/bNaJ6U3HrTE6kiium5/jqxDp/lSBg/T2qzGtTLxQLnkOWR/U04sT1NNoqbIHOQZpKKQ9Kom7YEjvTC1DUw0ABopKBSAdRSUUAaNtbtdXKQxkBnPBY4Ap13ZC3j3C7t5TnG2NyT/KptIYJq0LNggZ6/Q0ls8NxrlrugSOMzKGUdCM+9YTlJTfZK/wCZx1KtSNV2+GKv08/8jOFOrUgl+0XF0yxxmaIf6PGFGMZ5wO5xVwIvmWcl1DGJDDIzqy4yR0yKl1+Xdf1a5E8byOzj+Plf+n3MDFFa325/7NWbZF53m7d2wcLjpVxYrSG7vGSJnkDKVWIAkAjJIB96Uq7juv6/phPGygnzQ/Hs0v1Odqe2tXumbayRqoyzyHCipdTKG+YxRNEMDKsRnOPbpS2FlJeM4G4QoMyFRnj6etauf7vm2N5VV7D2rfLp6jDp0/2pYFCsWG5WDfKV9c+lJPp80LR4KSrIcI0TbgT6VoRu8lzPCY2hLWxjgRhg9QcfU4NOsx9jt7ZLr92zXO8BuqjbjP5/yrD2s1/kcjxVWKvpft30ev8AXmZ1xpstvEZC8UgU4cRvkofeqe2tdLeW1t71rhSoddi5H3juB4rMIrenNyvd3OuhUlNO7vbqvRMiIzTStS4pMVqdLISvFNKg1MVpNtBJCUqNhzVgrTCtAEG2l2mpQtLigZBiggmp9tBT2piK+2jaasbPal20AV9po2VY2Gl2H0pAVSlGyrJSgJTGQbaaUq1sppTNIRX20Yqby6XZVAQBaXbU3l0uykBW8ugJirOz2o2e1FwK+KUDNTbPajbincZDtpdhqYLTtlAitsxS7anKe1Gyi4FfbQFqbbQFoAhKUbKn20baAIdtG2p9tGw+lAEOKMVLsNKENMCHYaNhqfYaXYaAK+yjaam2GjbQBAQRRip9lGw0AQYoIqbZRsoAg20oWpdntS7KAIStJipylJsoAhwaMGpwntQE9qAIMUban2UoWgCvtNLtqfbRtoAgxSYqwUzSeV7UAVitL2qcx00x0AQ0YqXy6XZQBFilwalCUoT2oAhwaMVMUpNlAFfFFTlKb5ZoAipQKk8s04R0AQ4pcVLso2UAQ4oxU2yjZQBFRUmyl2UgIqSptlJspXGRUYqUR80uygCILRtNThOKCnFO4iDFJtqbZRtoYFcrTdtWClGz2ouBX2Gjaas7KPLouBX2Gjaan2GjYakCuVNN2nNWtlHlmmBV2nNKFqz5XtR5VAEKLT8VII8Uu2gCLBpwU08DNPC+1ADVUilqULSbTmkMZigrUoWjbQA1BgVJSAUtABSUtFACUh6U6mmgBhptKaTFMY2kpxpB1oEIaKVhRQIuk45FM75HWlpppFIVcqwZSQR0IpzO7NuZix9SaZS0WHZXuLnjrTlkZW3KxDeoPNNAzS4pBZMCcnJOT6mpI5ZI8+XIyZ67TimbRRtFFkyGk1ZjzI8jhndmYdCTQxZ2y7Fj6k5pAopwFCSGkl0FZ3YAM7MB0yelNOadtoIp2SKSS2I9tIV4p+KCM0ARYpQKftFG0UCGbaaUqbaKQqKARDtxRtqbYKNooBkO2l21NsFG0UEkO2kxUwUUbBQMhxSc1MVFJtFAEVKBTyoFAHNACYpNtS7RSbRQIi20u2pNoo2imMj20m2pgopdgoAiCcUbKsBRigqKQitsppQVYKgUBAaYyuFxTsVMUFN2igCMCgrU20U3aKBEBUCgKKe4xSLT6AAWl2U7vUg6UhkISl2VLig0CItlLs4qSl7UAQ7aNtPpQKBkWKNtPPWgUXEM24oxUjDpTaBjcUEVJtFIRii4EeKUCnU4DigCMikxUpUYpABQBHjFFPIpKdxDaKdRRcY0CjFPFJSuA3FLgU4U7AouBEVFBUYqXAowKaAg2UoSpdopyqKGIh8uk2+1T4plIZHs9qNntUopcUXEQ7KZtqwRTKaAjC0bakpRTAj20bBUmBS4pMCLbRtqUClwKLgQ+XQUxUtFICHbS7KlxS4GKQyDZRtqYjAptMBmOKNtPpKAG7BSFMVLSGgRFsox7VLSUAMC+1G32qQUUgGbaNgqSigCPYKQr7VLRigCILTttPxRQBGVpjIanpcD0oArKhqVRTyKQUxhRilxS0gG0U6koAbQaWg0ANpM0ppCaADNIxpKQnimAmaTNITSZpAGaKbmgHmgB5ooNFMD/9k=)

Figure 2: Trusov's code labelling the colours of each ball.

### TrackingSnookerBalls (Nirenshteyn, 2021)

Written in Python with OpenCV, Nirenshteyn’s code streams a video file, using hard-coded values to warp the perspective of each input frame to a top-down view. The green of the table is then discarded using a mask, leaving only the objects on or above the table. After applying the *findContours* function, the results are filtered based on their geometry to detect balls in the frame.

Having earlier converted the image to HSV, the colours are found using the average colour inside each ball’s contour and used to project the ball at its detected position onto a generated 2D top-down view. A frame of the result is shown in Figure 3.

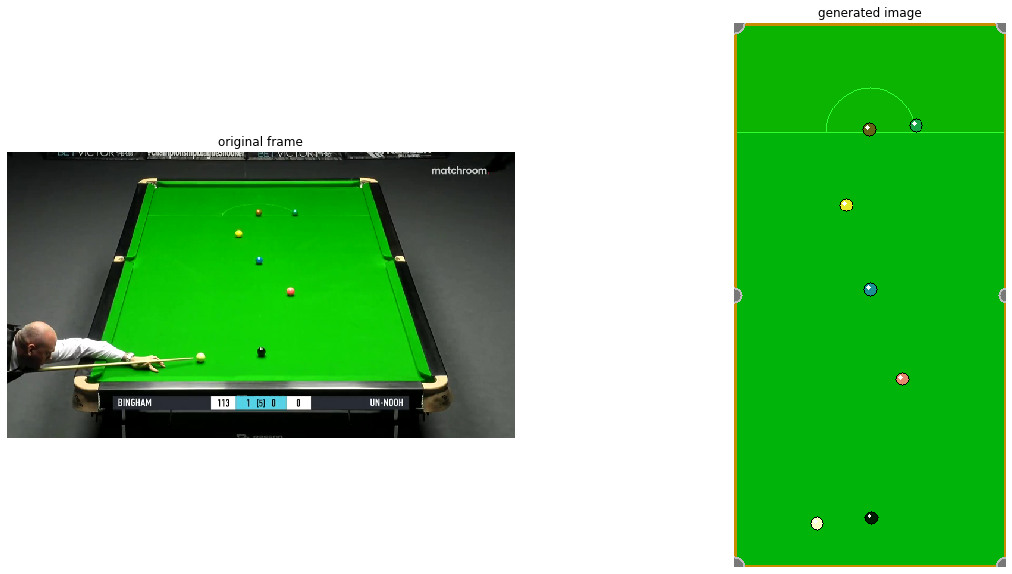


Figure 3: TrackingSnookerBalls generating a 2D view of the table.

### snooker-ball-tracker (Black, 2023)

A more complex project than any of the previously reviewed, Black set out to build a precursor to an automatic scoring system by first developing a ball tracking solution in Python, using OpenCV. The system features a GUI which displays a visual output of what the system is seeing, configurable options and debug and user info.

Graphical user interface

Description automatically generated

Figure 4: The snooker-ball-tracker GUI, showing options, output and what it sees.

Similarly to other projects, the table is masked, *findContours* is used, the contours are filtered, and the colours are detected in the HSV colour space.

This project goes further than others: there is some automatic potting detection and scoring functionality implemented but it remains a work in progress at the time of research and writing.

## Relevant Technologies

To determine the most applicable and effective technologies, research was conducted to determine available tools and their suitability for the project. Below is the resulting overview of tools found and conclusions drawn.

### Code Libraries

The aim being to develop a heavily computer vision-focused piece of software meant a library for this technology was chosen early in the project, with research being done into popular options. Multiple online lists (SuperAnnotate, 2021) (Boesch, The 12 Most Popular Computer Vision Tools in 2023, 2023) were reviewed and condensed into the shortlist below.

#### OpenCV

OpenCV is the “de facto standard tool for image processing” (Boesch, What is OpenCV? The Complete Guide, 2023), and its long-standing status as industry standard means there is a massive amount of information online about it, and well-maintained wrappers exist for languages beyond the officially supported C++, Python, Java and MATLAB interfaces (OpenCV, 2023), including languages such as JavaScript (OpenCV, 2023), C# (Emgu, 2022), and others.

Along with the wide variety of language support and online resources, OpenCV supports Windows, Linux, Android, and MacOS (OpenCV, 2023), making it a good choice for a proof-of-concept whose algorithms are likely to be ported to an unknown platform.

#### SimpleCV

As implied by its name, SimpleCV aims to provide a simpler approach to computer vision. It is not itself a library, but a framework wrapping a more simplistic interface around OpenCV and other libraries (SimpleCV, 2023). SimpleCV only supports Python, but does so with extensive documentation and tutorials.

Its simplicity makes it a tempting choice for developing a proof-of-concept, as it should allow for easier prototyping of algorithms.

#### BoofCV

(BoofCV, 2023) An open-source computer vision library available for Java.

Developed in Java and available for use in Java, Kotlin, Python, and Processing, BoofCV has extensive computer vision features that targets mainly higher-level processing (SuperAnnotate, 2021).

Given the time constraints of the project, the lesser popularity of BoofCV is worth considering as there are likely less learning resources and answered questions online than for other libraries.

#### TensorFlow

Though it is targeted at machine learning, this topic and computer vision are closely related, and TensorFlow provides computer vision tools for this reason. Providing official APIs for Python, C++, Java, and JavaScript (TensorFlow, 2021), and many other community-maintained language bindings being available, TensorFlow can be used on almost any development platform and has a huge amount of online support and resources available.

Due to the extensive learning resources and documentation available, TensorFlow is a tempting choice for a proof-of-concept app, but it is worth keeping in mind said resources will be mostly targeting machine learning as opposed to specifically computer vision.

#### OpenVINO

OpenVINO is a large framework providing algorithms from many different libraries such as TensorFlow, PyTorch, Caffe, and others. It aims to provide tools to leverage machine learning in vision, audio and language models, and targets C++ and Python (OpenVINO, 2023). Community-maintained APIs exist for languages such as Java (Kurtaev, 2023)

As OpenVINO is an attempt to bring together the most optimal algorithms from many different libraries, it is likely unnecessary for a technical proof-of-concept where optimisation is not of the utmost importance – a singular library may be preferable to keep the prototyping process simple.

### Languages

#### C++

While C++ may be famous for its efficiency[source], it is infamous for its difficulty to learn and unhelpfulness when developing software[source]. This would be a strong contender for a market-ready application, especially targeting efficiency to ensure viability on lower end hardware, as computer vision can be quite resource intensive.

However, given the aim of the project is to create a proof-of-concept, it may be wise to choose a language which allows for faster prototyping and ease of development.

#### Python

Python is known for being good for rapid prototyping (Zeller, 2020), with many developers using it as a platform for creating proof-of-concepts or rough implementations to better understand the required architecture or data structures to save rewriting more complex codebases later.

Even though there are personal gaps in knowledge and experience with Python development coming into the project, there is a huge wealth of relevant resources online and most Python concepts and libraries are designed ease of use in mind so these should not present a major issue.

#### C#

With Visual Studio as an IDE for writing C#, tools are provided to greatly ease the difficulty and speed of GUI creation in the form of its WPF designer. C# / .NET has extensive documentation and online support resources, and given its straightforward but modular approach to architecture[source or remove?], would make for simple prototyping.

Given the lack of official support from computer vision libraries however, it may not be the best choice as this is likely to result in less documentation and resources specific to the scope of this project, something that is not ideal when undertaking a large amount of learning before and during the development process.

#### HTML / JavaScript

HTML, backed up by JavaScript code, is the most popular development platform in the world (Stack Overflow, 2022), and potentially the most used, due to its deployment on the web and access via any user platform using any modern web browser.

HTML itself facilitates what is arguably one of the simplest GUI development processes, with positions of GUI elements often not even needing to be specified unless required, which may help avoid time wasted designing a GUI layout or getting tied up in UX/UI concerns developing what is a proof-of-concept where these concerns are mostly irrelevant.

JavaScript running in the backend of a HTML page enables not only dynamic control of the page (GUI) elements, but any other processing required, and the two work together seamlessly.

### Engineering Practices

# Design and Architecture

## Selected Technologies

HTML, JavaScript and OpenCV.

Following the research conducted into languages and computer vision libraries, the decision was made to use HTML and JavaScript as this will allow for very straightforward rapid prototyping of basic but intuitive UI via HTML and necessary algorithms via JavaScript. Paired with the choice of industry standard computer vision library OpenCV, which has an official JS interface and a good amount of online documentation and learning resources, the tools chosen should be more than sufficient for developing the proof-of-concept project.

Any other minor code libraries required may be used, if so, this will be specified where applicable in the Development section.

## Ideas and Approaches

Since the aim of the project is only to create a proof-of-concept app, there will not be much consideration given to fluent user experience design or stylisation of the user interface – so long as there is a reasonably intuitive method of operating the solution, this will be acceptable.

## Planned Architecture

Remember to make some diagrams maybe.

Though the application will be a website, it will be designed for local access – the use of HTML is only to provide a user interface and support JavaScript code running in the background.

Web-specific JavaScript features could be utilised but should not be at the core of the functionality as it should be possible to port the designed algorithms to another platform with ease.

# Development

Throughout the development process, a lot was learned, and the direction of the project was tweaked many times. This section will detail thoughts behind the technical implementation as well as any issues encountered, what was done to solve them, and their effect on the direction of the project.

## Terminology

There are some terms used in this section whose meanings may not be immediately clear, as they have been defined within the scope of the project.

**State:** The array of ball positions returned by *trackBalls()*, the function outlined in 4.4.1.

**Candidate state:** The state extracted from the current frame of source video.

**Current state:** The latest candidate state that is meaningfully different from the latest current state (to eliminate tracking inaccuracies)

**Last state:** The latest ‘end-of-shot’ state stored and displayed in the mini-canvases.

## Project Structure

The project is structured as a locally hosted website. Making use of a typical basic web folder structure – *html*, *js*, *media*, and *css* folders – the aim of the high-level architecture was to keep it simple to avoid wasting development time on something that is, in a proof-of-concept, mainly irrelevant.

Due to technical restrictions, which will be discussed in the Critical Evaluation, the project was hosted on a local server during development and testing. This was done using npm package *http-server* (Thornton, 2022).

The *index.html* landing page allows the user to choose an image or video file. Upon choice, the page redirects to */html/selectCorners.html* which asks the user to select the four corners of the table, as discussed in the next section. Finally, depending on whether an image or video was selected, the page redirects to */html/analyseVideo.html* or */html/analyseFrame.html*, and the deliverable processes the media.

## Perspective Warping

Before any detection takes place, there should be some ‘normalisation’ of the image to a corrected top-down view of the table; it will make detection far simpler if we can design detection and filtering algorithms with as many uncertainties as possible removed.

OpenCV provides a function to do this: *cv.warpPerspective*. It requires a transform calculated by cv.*getPerspectiveTransform*, into which we pass an array of user-selected corner values, retrieved on the *selectCorners* page using a click event listener recording the coordinates of four user clicks on a frame of the chosen video or image. Along with the corners of the output canvas, the transform is calculated and applied to warp the input frame appropriately:

A pool table with balls on it

Description automatically generated with medium confidence

Figure 5: A frame of example source video (Barton, 2021).

A picture containing pool table, pool ball, green, poolroom

Description automatically generated

Figure 6: The input frame warped to a top-down perspective.

This code is wrapped up in the *normaliseView* function and called on every new input frame. There are optional parameters to display the warped frame on a canvas using cv.*imShow*, and this functionality is used in the deliverable to show what the program is ‘looking at’ as it tracks.

## Tracking Balls

### Shapes

Many shape detection algorithms have been devised throughout the history of computer vision, and OpenCV implements some of the most widely used ones. These were investigated through online resources and the most appropriate seeming algorithms were trialled to ascertain their viability for the use case.

#### HoughCircles (Hough Transform)

Initially, the *cv.HoughCircles* function was used for ball detection, as it seemed an obvious choice out of the gate when reading about OpenCV’s capabilities. However, at the time it was difficult to produce acceptable results using the function so further research was done into other object detection algorithms which were then employed and developed further.

#### FindContours

Implementing a now famous algorithm published in 1985 (Suzuki & Abe, 1985), *cv.findContours* takes an input image and some basic parameters and returns the found contours. As we expect this algorithm to find objects that are not balls, we must filter the returned data. The method created for achieving this is as follows:

Get the contour’s *minAreaRect* – the bounding box around the contour. This bounding box will be rotated to fit the smallest size possible, but since we are looking for circular objects, we can forget about this angle as anything that fits our filter should do so with complete rotational symmetry.

Next, calculate the area and aspect ratio of the bounding box. The aspect ratio can be used to filter out any objects that are not circular or close to circular, as a perfect circle’s bounding box will have a 1:1 ratio – we simply discard objects that are not within a certain range around the perfect 1:1 value. Area is then used to further filter objects that may pass the aspect ratio check but are too big or small to be a ball.

The bounding boxes that successfully pass through this filter are added to an array and returned as the current state of the table.

### Colours

Now that we know the positions of the balls, we can use this information to get their colours from the perspective-corrected input image. The process is a simple one: a blank mask is created, and a small circle is drawn at the position of the ball. The mask is then applied to the input image and *cv.mean* is used to get the mean RGB value inside the masked area.

While all the example projects operated in the HSV colour space, this complication was avoided here as matching colours to the exact colours that the balls should be, was considered to be out of scope for the project. Instead, the mean RGB value that we just found is directly used as the fill colour for the relevant ball.

If colour matching were to be attempted, it would be advisable to convert the input image to either the HSV or L\*A\*B colour space, as these facilitate more accurate and reliable colour comparison.

## Basic Output – rendering the balls

As some form of output is necessary for debugging during development anyway, a function was created very early on to visually represent the positions of the balls. It simply draws a white circle at each ball’s position:

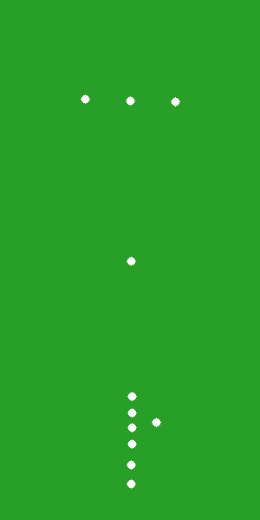


Figure 7: The warped input frame and rendered tracked balls.

Note that the size of the drawn balls is not dependent on the size of the detected ones, as this only opens the door to inconsistencies – we know exactly what size the balls should be and some balls, especially the green, may not have their entire area detected.

The output can be improved ­­by utilising the ball colour detection function and adding the table’s markings (white lines and dots) (Blue Moon Leisure, n.d.):

A picture containing background pattern

Description automatically generated

Figure 8: The tracked balls rendered in full colour with the table markings

The markings for the table are drawn statically; not detected from the tracked media. In Figure 8, some of the balls are visibly misaligned with the markings. This is mainly because of optical distortion in the source video, something that was not addressed in this project but will be discussed in the Critical Evaluation.

## Video

### Supporting video throughput

On the *analayseVideo* page, frame data is retrieved from a HTML video element via the OpenCV *VideoCapture* object’s *.read()* function. The source video is hidden via css and the perspective warped frames are displayed for debugging purposes.

### Detecting when a new shot has been played

#### Evaluating state difference

For each frame of video that is tracked, we want to determine if the balls have moved since the previous frame. This is an important piece of the functionality as we need this in order to detect when the balls have stopped moving, indicating they have come to a stop after a shot has been played, so that we can save their current state. To achieve this, a function called *compareStates* was created which returns a number representing the difference between two arrays of balls.

The function first attempts to discern which ball in the new frame correlates to which ball in the previous frame, which we do not explicitly know but can be determined with a high degree of certainty as we can expect the balls to not move much between two frames, especially when they are slowing to a stop, and this is the point in time that we are most interested in. This is done by iterating through each of the balls in one of the arrays and finding the ball closest to it in the other array; whenever a new closest ball is found, its index and distance from the ball are stored.

Finally, if there are any extra balls in the longer of the two arrays, the closest ball to each is double checked. This helps to reduce erroneously high values that may emerge from tracking errors (‘ghost balls’) or fast-moving balls.

If the *compareStates* function returns a value greater than a certain threshold – 5 was arbitrarily chosen after observing its output values for some time – then the candidate state is meaningfully different from the last known one, so we consider the balls to have moved, and we set the value of the current state to that of the candidate state.

#### Numbers of balls

In addition to calculating the difference in ball positions between states, the number of balls is compared. This is because we need to know if a ball was re-spotted, as if it was then we need to replace the previously stored state with the new one. For example, if the shot taken was a pot on the black ball, then the black ball will be placed back on its spot by the referee. However, it is possible that the system detected the end of the shot before the black ball was re-spotted, and we must update the stored state if this was the case.

#### Confidence values

Sometimes balls fail to be tracked for a frame or two, or, inversely, a video artefact will cause a temporary ‘ghost ball’. Additionally, the tracking algorithm does not have any temporal understanding, meaning each frame is tracked independent of information learned from the previous one, causing balls’ tracked positions to sometimes vary by a few pixels from frame to frame due to video artefacts.

Because the ball tracking functions do not provide perfect data, we must design the algorithms that rely on this data with this imperfection in mind. To account for this then, a series of values are used to express the code’s confidence that a change happened:

For example, if there is one less ball in the candidate state than the last recorded state, the so-called ‘confidence value’ for a ball being potted is incremented by 1, starting at 0. After each incrementation, its value is checked and if it reaches a pre-set threshold then the code assumes that a ball has indeed been potted.

These confidence values are used for the balls being still, a ball being potted, and a ball being re-spotted.

### Storing past states

When the above logic is applied, we occasionally find a new still state, whether a ball has been potted, replaced, or neither, we wish to store at least the most recent previous state for the purpose of replacing the balls if needed – this is the main aim of the program.

Some deliberation was given as to how to store past states, and which states need storing at all. To demonstrate functionality more clearly, and because the performance impact is minimal, the decision was made to retain a list of all previous states while the video is being analysed. The states themselves are already represented neatly as an array of objects and we simply push each new state to an array, giving us a record of all past states in the current video.

For the purpose of demonstration and debugging, each new state is rendered to a miniature canvas, and a message is displayed showing the program’s reasoning for adding or updating a state.

### ‘Rewinding’ to a previous shot on-demand

Thanks to the grid of all previous shots that the program shows, we can select any of the previous shots, for example using a click event listener on the mini canvases. A hotkey or button could also easily be added to recall the most recent saved table state; this would likely be the route taken in a market-ready application.

Accessing a past table state is trivial in the code thanks to the array of states, and a function already exists to render one of these states to a canvas complete with table markings, so creating a large display of a previous state is easily achieved:

***Code + screenshots go here.***

## Performance Considerations

Computer vision applications often require a lot of computational resources[source], as was in the Research section when reviewing similar projects which all struggled to run in real-time. Thankfully, due to the nature of this project, there are several liberties which can be taken which benefit performance:

#### Limiting colour detection

Detecting the colours of balls is expensive – the input frame is processed and scanned in several steps for each ball detected. Luckily, we do not need to detect the colour of the balls all of the time.

Given that the main aim of the project is to help replace the balls to a previous still position, we only need colour information when a still position is detected. This takes us from running colour detection once per frame to once every several seconds as the balls come to a stop at the end of a shot.

#### Real-time processing

Since we only care about accurate recording of position when the balls have come to a complete stop, the program does not actually need to keep up with all the input frames; when the balls are not moving the code will have enough time to realise this even if it is not running in real-time.

## Abstract Views

As discussed earlier, perspective warping is performed by the code to normalise every input frame to a top-down cropped view of just the bed of the table, as it helps to standardise the input data as much as possible. However, this method has its limitations:

A pool table in a room

Description automatically generated with medium confidence

Figure 9: A frame of video taken from a mobile phone placed as high as often possible in a casual setting.

A group of fish swimming in the water

Description automatically generated with low confidence

Figure 10: Figure 9 warped to a top-down view using the normaliseView() function.

If the angle of an image or video is too far from our desired top-down perspective, then a simple perspective warp results in an unusable output frame – too much visual information is lost and the warped image cannot be used to determine discern one ball from another, let alone any accurate positional data. To parse the visual data from this angle would require a different approach and heighten the need for correcting optical distortion, which is a problem not tackled in this project.

# Testing

As part of evaluating the success of the development process, thorough testing of the deliverable will be performed.

# Critical Evaluation

## Ethical Concerns

Due to the nature of the project and the approach taken in research and development, there was little in the way of ethical concerns, but an outline of considerations taken is as below.

### Research

No market research or user questionnaires were necessary in the research phase of the project as the deliverable was a technical proof-of-concept, so no ethical considerations were required.

### Development and testing

The main sample video used throughout testing and for demonstration in the report was taken from Steve Barton’s YouTube channel BartonSnooker. Explicit permission was obtained, as demonstrated in Appendix C. Thanks are extended to Steve for his enthusiastic cooperation.

As part of gathering further sample video for use during both the development and testing of the deliverable, an approximately 30-minute video was recorded of the developer and a family member playing a frame of snooker in the Lincoln Snooker Club. Explicit permission was granted by the owner of the club and any patrons who might have appeared in frame. Any frames or sections of video used in the report, or any demonstrations only include the developer and their family member.