

Project Description:

My project is a game played by 1 person in which the user is running away from a chaser, with a jetpack. He is constantly moving, but he has to get past barriers that will show up on the screen in front of him. There will be coins you have to collect while flying, and those will add to your score. There will be power ups, which can help you last longer in the game, or get a higher score. Once you hit a barrier, however, or get shot by your chaser from the back, it is game over. There will be achievements in a pause screen you can activate while playing, will have a start screen you enter before you start, and an end screen that has your final score and number of coins.

Competitive Analysis:

Everyone has 3 games they used to play obsessively when they were little. For me and I think for most other people, those were Temple Run, Subway Surfers, and Jetpack Joyride. Basically, my project combines these 3 games, but uses the main idea from Jetpack Joyride. These apps have many similar features: you only have one life, there are barriers that the user has to weave through or else they die, your score increases as you get farther into the game, you can pick up coins, and as the game goes on, the barriers arrive faster and the game becomes harder to get out of. The difference between jetpack joyride and the other games is that it doesn't have someone chasing you. I added this, and as a distinct feature that is not in either game, is that the person chasing you can shoot you from the back, so you will have to watch both directions, which is different from all games, yet brings them together.

Structural Plan:

This will be a modal app, with multiple different classes inside each mode. This modalApp class will store a "global" variable that tracks the number of coins and points the user has at each moment. Let's start with the Start Screen which will be in a separate file. This will be a mode with multiple buttons, that will display the name "ChaseDown" and some kind of background image. Each button will have a specific purpose: such as starting the game or checking the current achievements. There will mostly be onClick functions and some button classes if I need them. Then, the actual game mode screen. This will be another mode, and will implement many different classes because there will be multiple things on the screen. These classes will all be implemented in different files so they can be tested separately in different files. The classes will be the Steve class, which is the name of the user. This class will have a render and move function that will be triggered in the GameMode class by different keys being pressed. The other classes will be the Chaser class, which will basically follow the user Steve, and lastly a Barrier class to create different barriers on the screen. The next mode will be the PauseScreen Mode which will have different achievements that the user can try to obtain in their run. They will also be able to see their score and number of coins. Lastly, after you die in the gameMode, there will be a GameOverMode, which will display the achievements achieved, the user's score, and the user's number of collected coins.

Algorithmic Plan:

The Algorithmic Plan to create the most complex part of the code is creating the different barriers as well as the power ups.

1. Get State of the Game
 - a. In order to place barriers, you will need to know where the user is at that specific moment
 - b. We will need to check if the user is touching the barrier or powerUp in order to change or stop the game
 - c. You will also need to know the app size, because you will need constraints of where to put the barriers and Power Ups
2. Create Barrier/Power Up
 - a. We will create a barrier/power up based on how many seconds/timer fired calls have passed in Mode
 - b. Their location will have to be adjusted based on scrollX
 - c. Each power up will have a specific purpose that will change the state of the users movements and actions on the screen
 - d. The barrier must have a function to check if the user is between the two end points to check if the game is over
3. Render the Features
 - a. Based on the constraints, we will make the Barrier/PowerUp and add it to the list of barriers and power ups inside of the mode
 - b. In redrawAll, we will loop through all the features and render them onto the screen using their render own function

Timeline Plan:

Monday: Skeleton Code Complete

Tuesday: Graphic Images collected, StartScreen and PauseScreen finished

Wednesday:

- Before Meeting: Can Move User and Chaser, can change background while playing and keep track of score
- After Meeting: Make Barriers

Thursday:

- Make Coins and collect Coins
- Finish Game Over screen

Friday:

- Finish Demo and Practice

Version Control Plan:


I will be using GitHub to upload each version of my project, and store them there if I corrupt my files or need to go back to an older version if I need to start over. It is super easy to pull and push to git, and will provide easy access from all devices.


🔗 master ▾

ChaseDown / ChaseDown /


Go to file

Add file ▾

 abhisaxena3035 Add files via upload


3891c8e 36 seconds ago  History

..

 ChaseDown.py


Add files via upload

35 seconds ago

 Chaser.py


Add files via upload

35 seconds ago

 GameScreen.py


Add files via upload

35 seconds ago

 StartScreen.py

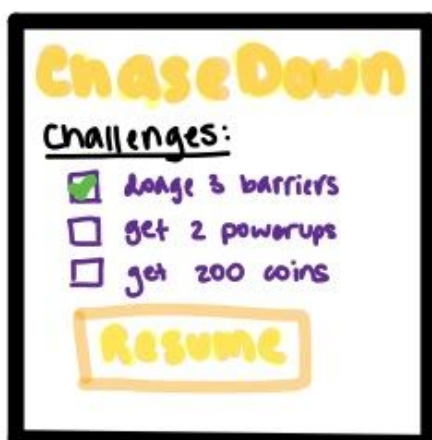
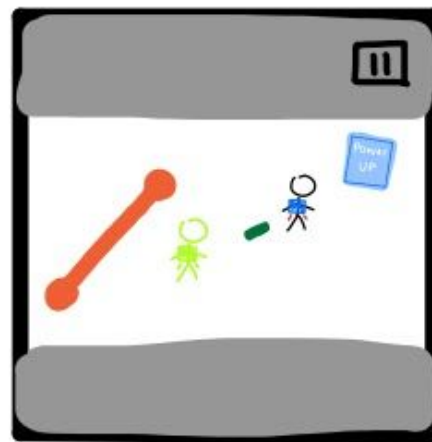
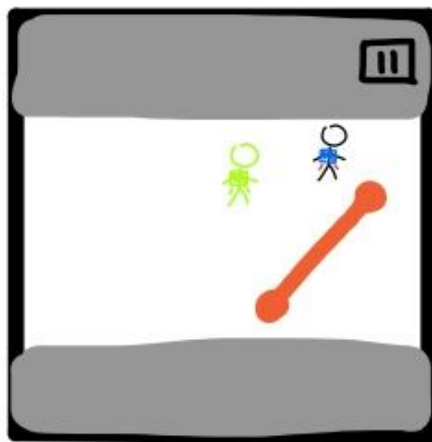
Add files via upload

35 seconds ago

 User.py

Add files via upload

35 seconds ago



TP 2 Update:

- The algorithmic complexity of the program comes from recursive backtracking when trying to place the barriers
- There are now different orientations and sizes of the barriers
- All other aspects of the code stayed the same