Walker Trelease & Camille Girard

# runRun

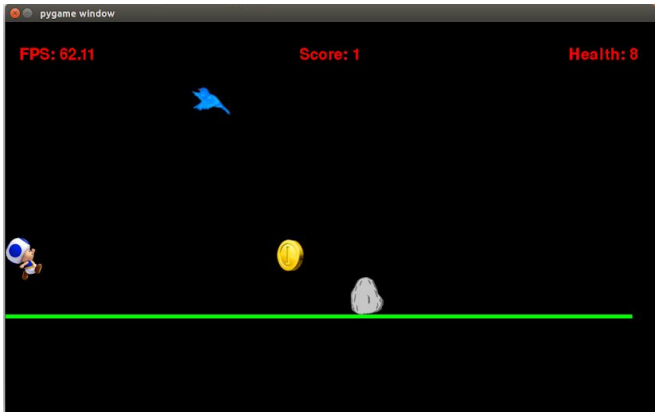
**Project Overview:**

We created a 2D side-scroller game for an individual player on a keyboard on a computer. It features a running character on an uneven ground controlled by the arrow keys and the space bar, coins that count for points, and birds and rocks that take away health points.

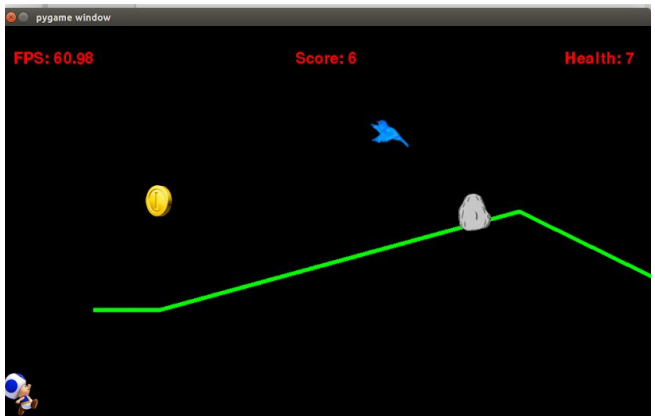
**Results**:

For the final results we had a full game with lives and points as well as a game over screen which can make the game go again if the player presses p. This game over screen appears if the player falls in a hole in the ground or if the player hits enough obstacles to lose the ten lives they begin with. This health is shown on the top right of the screen and is damaged by one every time you hit an obstacle. The score comes from the collection of coins - each worth one- and will appear on the screen as you play and on the final screen.


The player is able to move forward and backwards as well as jump, and is continuous if the key is held down. The coin moves up and down in place and will disappear if the player collects it. The rock will spawn randomly and will be static on the ground. The bird with fly up and down within a set window with varying speeds for each bird. If the player hits a rock or a bird it will deflect to the side but the rock and bird will remain there.




The ground moves under the player to give the impression of the player running, but the player can also move back and forth. The player cannot go all the way to the edge of the screen since this would not allow them to see the obstacles coming. The ground is spawned in a way that there are gaps, which the player must leap over or have a game over, and also has varying heights throughout the game combined with slopes or flat sections.

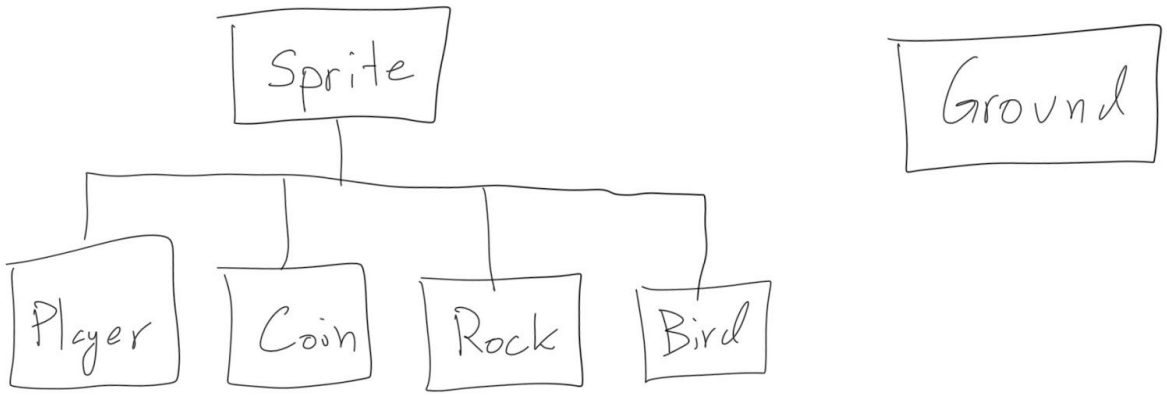



**Implementation**:

Our game consists of five main classes from which many objects are constantly being generated from and updating as the game runs. The game itself is run inside the main class, which after initializing various game parameters enters the main game loop. The loop first tracks time and checks for player input, and then checks for game events and updates all objects. Game events include the player colliding with any of the other game objects, and appropriate action being taken. The game updating starts by advancing all current items to the next point in time, and then adding new items as appropriate. We have a game speed parameter, which when multiplied by the time since the last frame tells us how far to advance the game, and the ground and all objects move left that distance. Coins and Birds also update their heights, and the player speed and location will update based on user inputs, collisions, etc. Once all of this has been done the next frame is drawn and the process repeats until a game end event is

triggered and GAME OVER and final score is displayed, at which point the user can exit or choose to play again.

A UML class diagram is listed below, it is relatively straightforward with most of our objects inheriting from the pygame.Sprite class with the exception of ground which is its own class. Coin, Rock, and Bird are very similar and probably could have all inherited from a Item() class which itself inherits from the Sprite class, but we were not familiar enough with the intricacies of inheritance to fully implement that design.




One design decision we made was how to create the ground, specifically if it should be a pygame.Sprite() object or a brand new class. We went with a new class because it can be represented pretty simply as a list of heights, and this seemed like it would be easier to detect collisions as it is fairly easy to get ground height at the x location of an object and then text if the object is above or below that.

**Reflection**:

For our team process we each worked and kept in touch with each other, giving out ideas of what to do next and getting pieces done independently. We mostly operated by time splitting, and did not have any merge conflicts or bugs and not really any problems. We built a lot of skills in pygame which is applicable in the future and built an interesting game which can be improved and built on even more.

Our project was well scoped, and we could keep adding and making it better as much as we could which worked well. We started with a very simple version and were able to add to it as we could and in ways we wanted to, which made it easy to scope and stay interested in. The game could definitely be improved, such as making better reactions from hitting obstacles, which would be a project for future exploration. It would have been nice to maybe layout what each aspect needed and this could have also lead to having an item class since the three pieces other than the player were very similar, which would have made the code nicer.