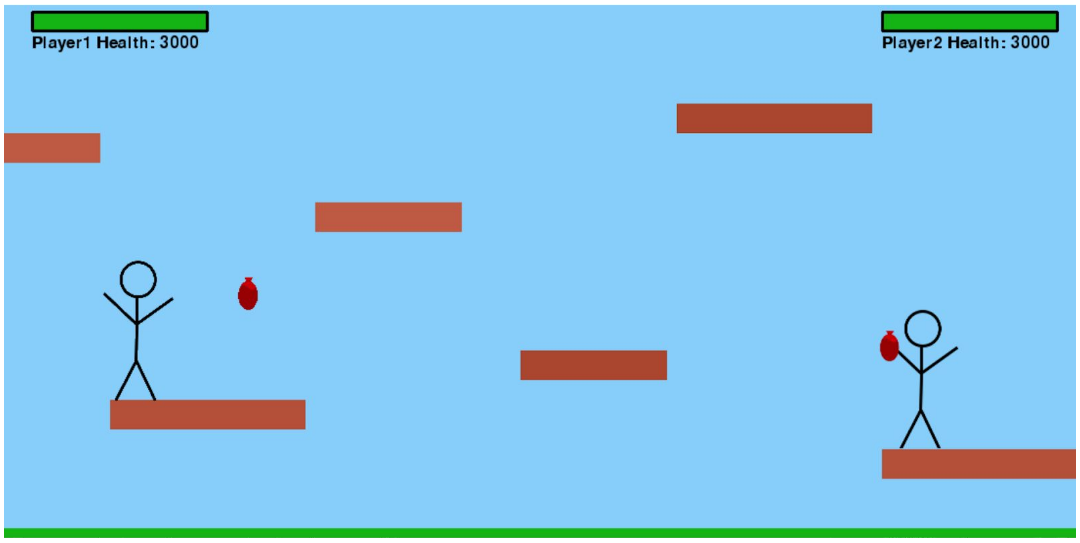
Water Warriors



Final Catapult Report - Team 19

Water Wars

Claire Umpleby, North Shore Country Day School

Grace Keeton, Tri-West High School

1 August 2019

Project Summary

The game was coded in Python in a Street Fighter style based off one of the most popular Operation Catapult activities—water wars. Water Wars is a two player game where the opponents throw water balloons at each other and take damage when hit. The first player to run out of health loses the game. To make gameplay more difficult, the map consists of various platforms that players can jump on to avoid the balloons.

Factors that posed challenges during the coding process mostly involved making the characters interact correctly with their environment. The primary challenge was to make it so players would land on the platforms below them and fall directly to the ground when stepping off a platform as opposed to stopping at the height of another platform. We also spent time making the physics of our game reflect real world physics. This included making the characters arc after jumping instead of falling straight down which meant implementing velocity vectors and gravity into the code.

In order to make the game more difficult, the team decided to add a three second delay between each fire of the balloon. The implementation of the delay required a second image for each player in which they were not holding their balloon and additional code to coordinate the sprite and ability to fire. The aforementioned coding proved to be quite technically challenging, because the code had to be structured in a way that did not interfere with the functions already built into the game.

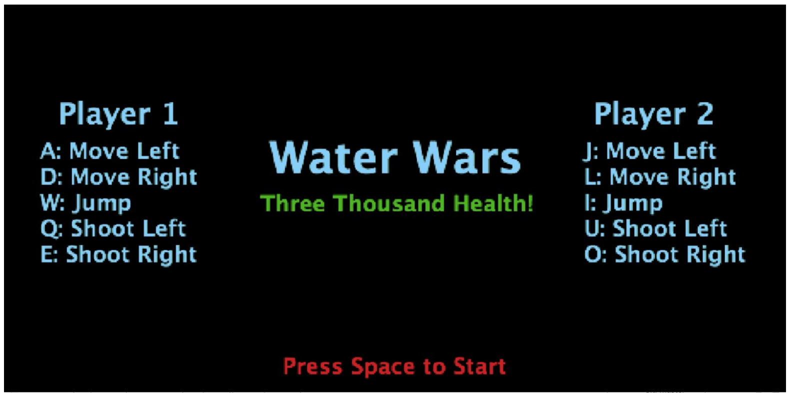
With over two hundred fifty lines of code in the final version, staying organized became a major challenge. The large number of classes and functions made revisions and troubleshooting significantly more difficult. To keep the program organized and clear to read, the team incorporated explanatory comments throughout the code.

Working on this project drastically improved the team's understanding of the fundamental concepts and skills of computer science. It also provided a working knowledge of the Python language. The Water Warriors learned to effectively work together and created a product to be proud of.

Introduction

The coding language Python was used to code the Street Fighter style game, Water Wars. The name and gameplay is based on one of the most popular Operation Catapult activities—water wars—at which there were many water balloon fights. In the Python game, two players face off, throwing water balloons at each other to decrease their opponent's health. The first player to run out of health—be hit three times—loses the game. The gameplay is made a little more challenging with the platforms players can jump onto.

How to Play


Player 1

A: Move Left
D: Move Right
W: Jump
Q: Shoot Left
E: Shoot Right

Water Wars

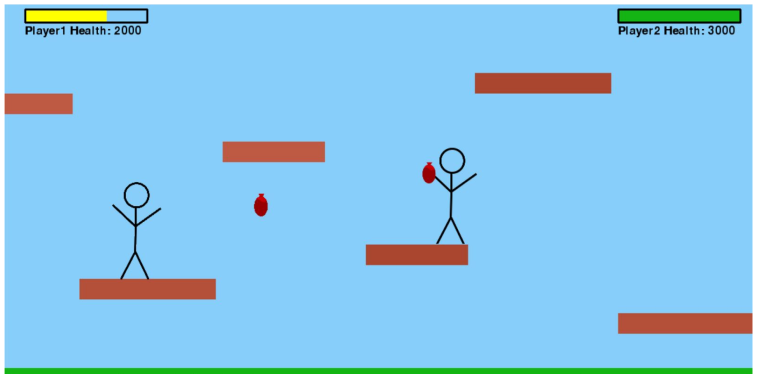
Three Thousand Health!

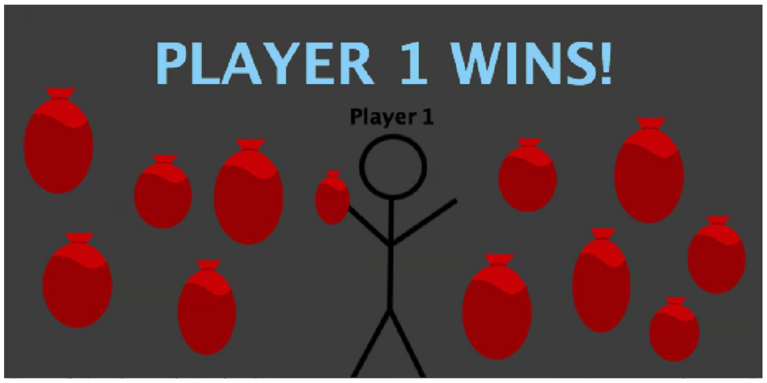
Player 2

J: Move Left
L: Move Right
I: Jump
U: Shoot Left
O: Shoot Right

Press Space to Start

The goal of the game is to shoot your opponent with balloons until they run out of health. Each player has a set of keys for their basic movement as shown in the start screen shown above. Player 1 uses A to move left, D to move right, W to jump, Q to shoot left, and E to shoot right. Player 2 uses J to move left, L to move right, I to jump, U to shoot left, and O to shoot right. Gameplay is made more interesting and difficult, because players can also jump onto platforms to help them dodge balloons as depicted in the image below. In order to win, players need to hit their opponent with a balloon three times after which the end screen below will appear.





About Python

The Python project was started by Guido van Rossum in the late 1980s, and was developed through collaboration with his colleagues at the National Research Institute of Mathematics and Computer Science (Wolfe, 2019). Despite the fact that the logo is an image of two snakes, the name Python does not come from the species of snake. Instead, it is inspired by Monty Python, a british comedy group (Patrick, 2019). It was designed to be a successor to the ABC language, which was developed a few years earlier in the Netherlands (Wolfe, 2019). The goal of this new language was to have more capabilities and complexity than ABC, but to still maintain similar syntax and relative ease of use (Patrick, 2019).

Since version 0.9.0 was released in 1991, there have been numerous versions and updates. The most recent is 3.7.4 which was released on July 8, 2019 (Patrick, 2019). Today, Python is used by many major corporations. Companies such as Google, IBM, and Dropbox have implemented it as their main coding language due to its simplicity, efficiency, and ease of use (Patrick, 2019). The programming language also has a large fanbase of individual users, who even appointed Guido van Rossum to the position of Python's 'Benevolent Dictator for Life' (Patrick, 2019). It is common for programmers to be familiar with multiple languages. In fact, only 9 percent of Python programmers report it is the only language they use. However, Python is so popular amongst its users that nearly four out of five python developers say it is their main language (Jodlowska, 2018).

Coding Process

Before starting the final project, students spent 4.5 days learning the basics Python language. This crash course covered variables, functions, conditionals, loops, and classes. Students learned and applied this through lectures, pair programming, and coding smaller practice games.

After brainstorming game ideas individually, groups were formed and project ideas were shared and solidified. The Water Warriors used a decision matrix to compare their ideas. After the decision to make Water Wars was made, the team planned the basic functions for the game and potential future improvements. This included making sketches of the graphics and charts to visualize the flow of the gameplay.

Once the plan was solidified, the team began coding, starting with the framework for the game—a main function and while True loop. This basic framework was used to initialize and implement imports from pygame, sys, and time. The first additions to the function and loop created the clock and screen and allowed the user to exit the program without having to do it manually.

The main framework for the game was organized into classes with functions that are called in the aforementioned main function and while True loop. This started with more planning to organize what classes there needed to be and what functions they should include. The five classes—player, balloon, platforms, health, and display—and seven functions—draw, move, jump, land_plat_below, fire, hit_by, remove_exploded_balloons, and damage—were then incrementally added to the code.

Example Code

```
1 def jump(self):
2   if self.jumping == True:
3       self.current_y_velocity += self.gravity
4       self.move(0, -self.current_y_velocity)
5   self.land_plat_below()
```

Within the player class, there is a function to jump that is called when either player presses their respective jump key. This function is interesting because it imitates the physics of the real world. If the player is jumping when the function is called, it adds gravity, -1, to the player's current velocity in the y direction and changes its y position based on that new velocity. This allows the player to realistically move through space during a jump by decreasing in velocity until the top of the jump then increasing in velocity as the sprite falls, imitating gravity. On line 5, the land_plat_below function is called, which will detect whether or not there is a platform below the player and allow them to land at the correct height.

```
1 def land_plat_below(self):
2   y_below = 384
3   for i in range(len(self.platforms)):
4       platform = self.platforms[i]
5       if self.x >= platform.x - 45 and self.x <= platform.width + platform.x - 40:
6           y_below = platform.y - 145
7           break
8   if self.current_y_velocity < 0 and self.jumping:
9       if self.y + 146 >= platform.y:
10          self.y = y_below
11          self.jumping = False
12          self.current_y_velocity = self.original_y_velocity
13  if not self.jumping:
14      if y_below < self.y:
15          y_below = 384
16      if y_below > self.y:
17          self.move(0, 10)
```

One of the most complex pieces of code in the game is the land_plat_below function. In order to have the sprite land on a platform, it must detect if there is a platform below it and land on it. The maximum y value for any sprite is 384 because of the grass platform at the bottom of the screen. The variable y_below is defined as 384 to represent the ground. The next line sets the list of platforms as a single variable, platform. The function then determines if the sprite is horizontally above a platform for which it will return true. The subtraction of 45 and 40 pixels compensates for the width of the player image so that players can stand halfway off the edge of platforms. If the if statement returns true, then y_below is set to the height of the platform minus the height of the sprite, so the player will stand on the platform. The break instructs the loop to stop running after one of the platforms returns true, because it is not possible for a sprite to be in the same x location as multiple platforms.

The if statement on line 8 checks if the player is moving downward during a jump and has the potential to land on a platform. The next if statement checks if the player is above a platform, accounting for the height of the player. If the statement returns true, it sets the new ground to the y position of the platform. When this occurs, jumping is also set to false and velocity is returned to zero so that the sprite will stop falling.

Lastly, the if not statement on line 13 accounts for the scenario where a player walks off the edge of a platform instead of jumping. Because the previous functions set y_below to the height of the platforms the player lands on, lines 14 and 15 set y_below back to the height of the ground if y_below is above player, as would be the case if the player walked off a platform. The if statement on line 16 allows the player to move downward vertically at a rate of ten pixels for every update of the screen until the player's y is equal to y_below and has therefore reached the ground.

Problems Encountered

The largest problem the team encountered was implementing the physics of the real world into the game so that players would interact correctly with their environment. This included the ability to land on platforms, fall to the ground when walking off platforms, and jump in a parabolic path due to gravity.

The execution of a three second delay between firing proved to be another difficult problem. This seemingly simple addition to make the game more difficult required a second image for both players so they could be seen without a balloon. The addition required approximately 15 additional lines of code to coordinate the sprites and ability to fire.

The organization of this program with over 250 lines of code proved to be quite challenging as well. Revision and troubleshooting was a large problem the team encountered because of the size of the program and the duration of the work. To make sections of the code easier to find, read, and interpret, the team included explanatory comments throughout the code for future reference.

Reflection

Grace Keeton:

Our team functioned extremely well without challenges. We avoided this by sharing the keyboard and other responsibilities evenly and delegating work based on our individual interests. I had more experience with Python and coding in general to contribute to the project, but Claire has a Macbook and experience using its graphic functions. This made her extremely valuable in designing the game's graphics. The most technical challenge we faced as a team was implementing the land_plat_below function. It took over eight hours of the team's work with guidance from the teacher's assistants before it functioned properly. To overcome this challenge without falling behind elsewhere, I worked with the teacher's assistants on the function while Claire progressed the rest of the project. Both of us came in with ambitious mindsets and were excited to work together which was the main contribution to the success of our project.

Claire Umpleby:

During the first days of catapult when we were learning the basics of python, we got to work in pairs with various members of the computer science group. So when the time to choose teams for the game project, Grace and I already had some idea that we could work well together. This turned out to be true. At first I was a bit intimidated to code significant features of the game, because Grace is more experienced that I am. We ended up dividing up tasks in a way that allowed us both to work at our own pace on the parts of the code that interested us. Grace had the official code on her computer, so she would send me a copy of the most recent version, and then we would discuss what we wanted to get done that day, and split up jobs so that we could both work simultaneously. Both of us were genuinely excited about designing and creating the game which made it easy for us to work together towards our common goal. It also meant both of us were willing to work hard even when it got frustrating. After coming into these two weeks knowing absolutely nothing about Python, I am surprised by how much I am now able to do. I really enjoyed working with Python, and am definitely planning on either adding to this project or starting something new after catapult is finished.

References

[1] Wolfe, J. (2019). *A Brief History of Python*. [online] Medium. Available at: https://medium.com/@johnwolfe820/a-brief-history-of-python-ca2fa1f2e99e [Accessed 1 Aug. 2019].

[2] Patrick, S. (2019). *History of Python*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/history-of-python/ [Accessed 1 Aug. 2019].

[3] Jodlowska, E. (2018). *By the numbers: Python community trends in 2017/2018*. [online] Opensource.com.Available at: https://opensource.com/article/18/5/numbers-python-community-trends [Accessed 1 Aug. 2019].