Mini-Project 4
SoftDes SP18
March 6, 2018
Alex Frye | Sebastian Calvo

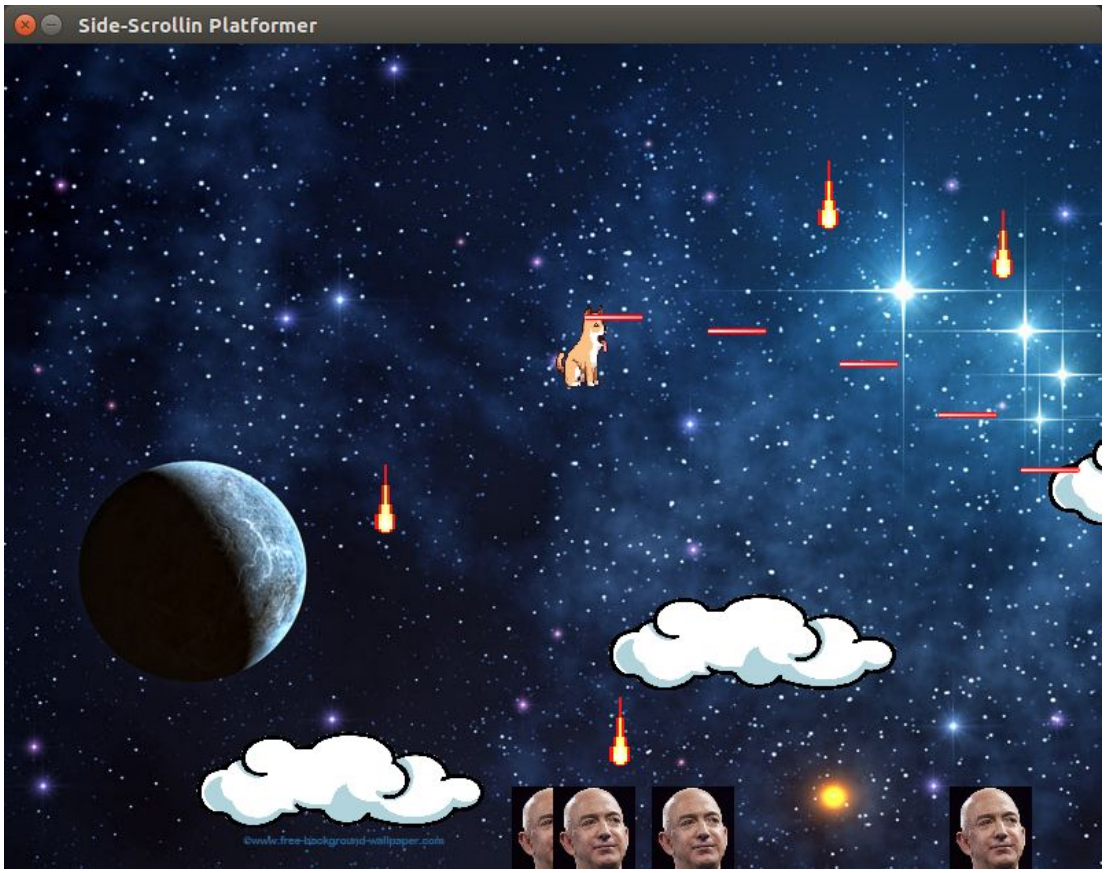
Project 4
Write-up & Reflection

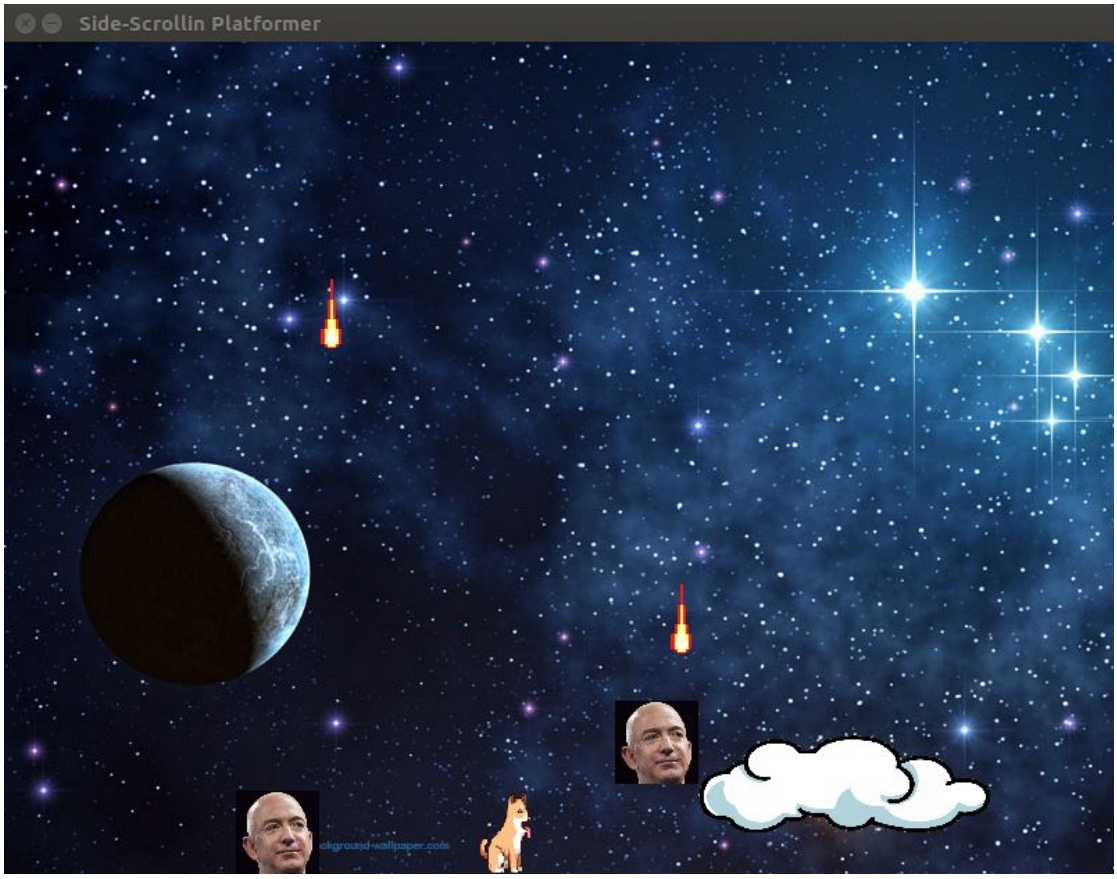
Project Overview:

The goal of this project was to create a computer game. More specifically a side scrolling platformer. Sub goals included creating a user controlled character, obstacles, and enemies. After meeting these sub-goals, the plan was to add additional features like a health system and aesthetic changes.

Results:

Our final game meets all of these goals. Our final product is a game where a user controlled player traverses a pre-created level. The player must dodge falling meteorites to stay alive and shoot lasers at jeff bezos heads in order to score points. After meeting the requirements of our minimal viable product we were even able to look into aesthetic changes.



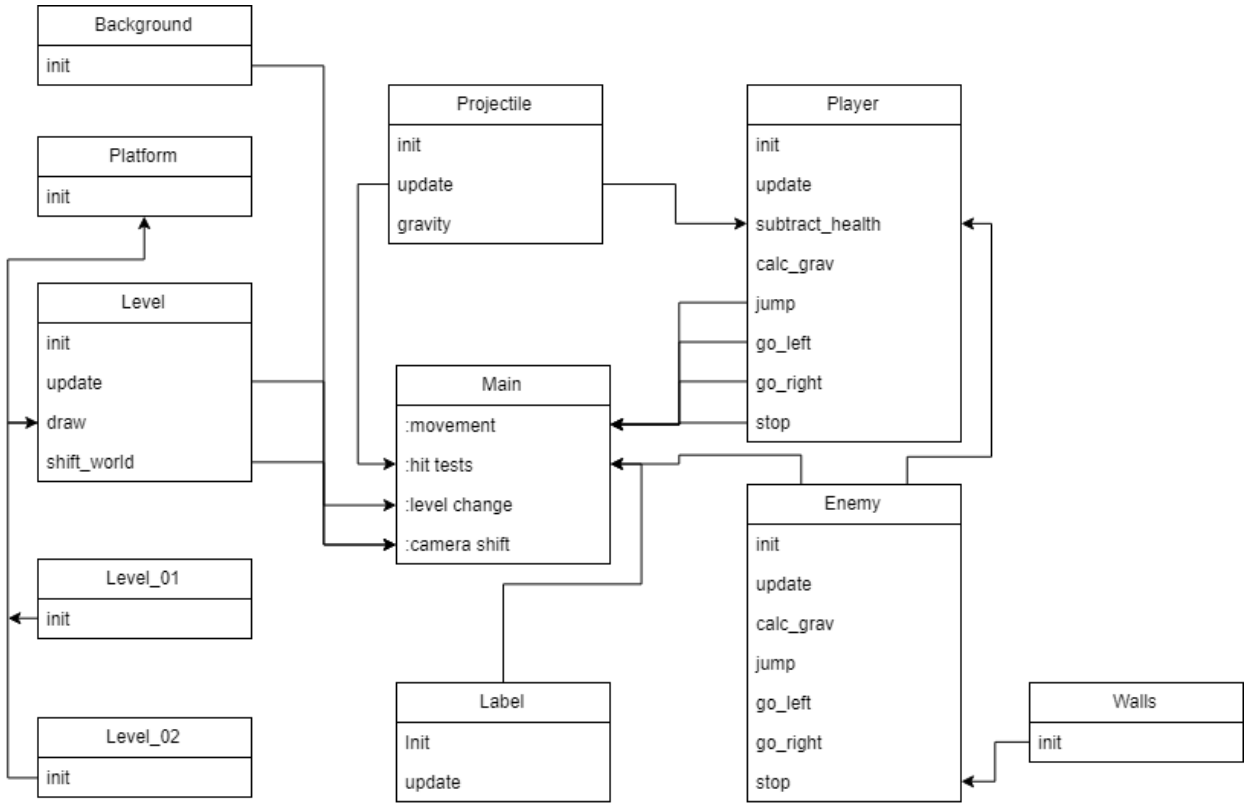


The game has a side scrolling feature that allows the player to continue moving infinitely to the left or right, until they meet the wall feature that is put into each level allowing each level of the game to be arbitrarily large.

Implementation:

The game architecture relies on ten classes, all of which represent a different sprite (or combine several instances of a sprite into another object) within the game. Each class is comprised of both the sptrits' rendered form and of its actions/attributes. In order to allow both partners to effectively work on different parts of the project in parallel, every class was placed in its own file and imported where needed. Each class is called in the main() function (main.py), which instantiates each sprite and runs a while loop to receive keyboard commands and update the game as necessary. Fortunately, PyGame is well suited for object oriented programming, so the impact of our lack experience was reduced. Additionally, since each class represented additional game functionality, our progressive implementation strategy was easy to execute.

The below UML diagram depicts the classes and functions that comprise our game. The Player class defines a user controlled sprite, while the Enemy is capable of navigating the game on their own. Projectiles can be called to fall as meteors or fire forward from the player as bullets (falling projectiles are in their own sprite list, and have different effects from bullets on collision). Each Level calls on Level_01 or Level_02 to define the location of each platform within the level. Platform is called to instantiate a block (appears as a cloud) within the Level, at the coordinates defined by Level_01 or Level_02. Label was intended to provide a blank text block where location, color, size, and text could be defined to display anything from gameplay instruction to remaining health, however, rendering it has proven difficult and it won't appear in this version of the game.



UML diagram of major classes and methods

Reflection:

As far as creating the game goes, it went very smoothly. We divided the work by class, each implementing a different part of the game before combining them together. Since our project had so many classes this was a much more effective approach than pair programming would have been. This also allowed for easier testing of our code, as we could each test our own parts before combining them into the final product.

We had a well scoped project since the complexity of the game could be easily scaled up or down depending on how the process of creating it was going. We found that starting with a simple minimum viable product, and scaling up in complexity as time allowed to be an effective method.

Additional Comments:

-Sebastian: I have always wanted to learn how to create video games and this project has made me even more excited about the topic. I will definitely use pygame again in the future.

-Alex: Working with PyGame was a great introduction to object-oriented programming and I appreciated how effectively we were able to break up a project into manageable modules. The modular design that was implemented is something I'd like to integrate into future projects.