Ben van Zyll
CIS 1051
Due: Feb. 21, 2023

Final Project: Software Development Plan

Introduction

Introducing *The Space Race*. This game will be a simple, arcade-style single-player web application developed using Pygame. The objective of the game is for the player to control a spaceship through space, avoiding obstacles in the form of asteroids, while collecting planets to increase their score, called “Planet’s Visited”. The player starts with three lives that they can lose by colliding with the asteroids. This project was selected as it offers a similar complexity to the snake game covered in class, making it an ideal game to create independently. Moreover, the style of game allows the use of many of the Python concepts covered in class, thanks to many features available within the Pygame module. The development process will involve the use of the Python3 programming language, GitHub for version control, and a Dockerized container to enable the game to be played across various machines. This software development plan will outline the steps required to design, implement, deploy, and maintain the Space Race game.

Requirements

The Space Race game will have several functionalities and features to provide an engaging experience to the players. The game's graphical user interface will be designed using Pygame, and the player will control the spaceship using the arrow keys. The game's functionalities include:

- ***Movement***: The player will be able to move their spaceship up and down using the arrow keys as controls.

- ***Obstacles***: The game will generate obstacles in the form of asteroids that move towards the player's spaceship at a constant speed. Perhaps for “leveling” the game, asteroids could speed up after incremental point or time thresholds.
- ***Planets***: Planets will appear randomly on the screen and can be collected (“visited”) by the player's spaceship by colliding with them.
- ***Lives***: The player will start with 3 lives, and each time they collide with an asteroid, they will lose a life.
- ***Scoring***: The game will have a scoring system, and the player's score will be calculated based on the number of planets visited.
- ***Timer***: The game will include a timer, and the difficulty will increase incrementally with time thresholds.
- ***Start:*** The player will also have the option to start a new game, pause, resume, or quit by clicking certain buttons.
- ***Save:*** The player will have the option to save the current state, quit, and later resume the game at the saved point.

The hardware requirements for the Space Race game will be minimal, and the game will be able to run on any computer with a modern browser. The software requirements for the development environment include an IDE (whether that be a web IDE or something like VSCode), a Dockerized container, GitHub for version control, and Pygame modules and functions, such as Pygame.display, Pygame.event, and Pygame.time. The game will be deployed and packaged using Pygbag.

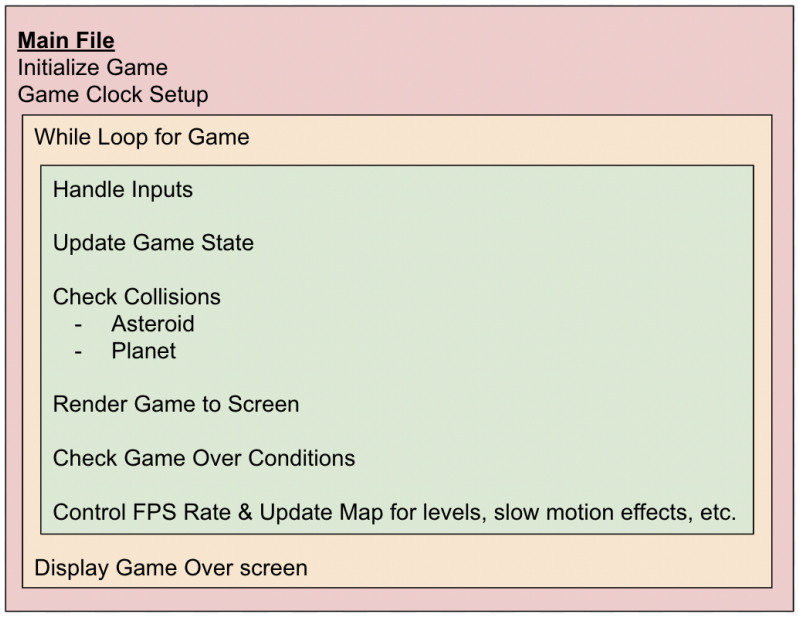
Design

The Space Race game will have a game loop that manages the game's flow of activities, including input handling, object updating, and rendering. The game loop will continuously run until the game is over. The game flow will include the spawning of asteroids and planets, the movement of the player's spaceship and asteroids, collision detection between the player's spaceship and obstacles, and updating the score and timer. Finally, as time passes, we will update our maps to make the game more difficult, in a “leveling” fashion. The input handling will involve reacting to the arrow keys for player movement and responding accordingly. Additionally, the game will have a sprite class for each object in the game, such as the spaceship, asteroids, and planets. Each sprite class will handle the object's movement, collision detection, and rendering.

The Space Race game will use various data structures to represent the game objects.

- The player's spaceship will be represented as a sprite, with its position and movement controlled by a Pygame “Rect” object.
- The asteroids will be represented as sprite classes, and their position and movement will also be controlled by Pygame Rect objects as well.
- The planets will be generated randomly on the screen and will also be represented as sprite classes.

To detect collisions, the game will use Pygame sprite collision detection, which works by checking for overlaps between the Rect objects of the sprites. The game will also use Python lists to store and manage the sprites in the game, which will enable efficient collision detection and rendering. Overall, the use of these data structures will enable efficient and responsive gameplay as well as the possibility to save game sessions.



Implementation

The implementation of Space Race will be broken down into several components, each of which will be developed step-by-step.

- The first component will be the player's spaceship, which will be implemented as a sprite class with its own movement and collision detection methods.
- The next component will be the asteroids, which will also be implemented as a sprite class with their own movement and collision detection methods.
- The third component will be the planets, which will be implemented as another sprite class, and will also have its own methods for spawning and collision detection.
- Finally, the functionality game will implement the game loop and timer, which will handle the overall flow of the game.

To ensure that the different components of the game interact seamlessly, the game will use Pygame's sprite groups to manage the sprites and Pygame's event handling system to handle user input. The game will also incorporate a Game class to encapsulate the game loop and timer functionality, which will use the individual sprite classes and event handlers to create a cohesive game experience.

To ensure readability and maintainability of the code, the game will follow coding conventions such as docstrings to describe interfaces, and docstrings at the top of each file to explain the classes and modules created in that file. The code will also adhere to common guidelines for naming conventions, line length, and other style considerations.

Testing will be an important part of the development process, and use unit tests to ensure that each component of the game is working as expected. For example, unit tests will be written to verify that the collision detection algorithms are detecting collisions correctly, and that the player's spaceship is moving and responding to user input as expected.

Deployment

There are two options for deployment: an offline and an online. In terms of offline deployment of the game, the first step will be to create an executable file that can be run on the target system. Packaging the game will involve creating a directory for the game's assets, including images, sounds, and fonts. The packaged game will then be installed onto the target system using an installer program. During deployment, there may be issues with compatibility with different operating systems, as Pygame is not fully compatible with all operating systems. To mitigate these issues, the game will be run in a Docker container, which can be built with a Dockerfile and then run on any system that has Docker installed. The online deployment option is to use 3rd party software such as a Docker container, which comes with Pygbag — containing a cross-platform solution for packaging and deploying Pygame applications.

Maintenance

Maintaining the game will be crucial for keeping it up to date and running smoothly over time. One of the main procedures for maintaining the game will be fixing bugs. Any bugs discovered will be documented and prioritized based on their impact and severity. Once

prioritized, the bugs will be addressed in the order of importance, and fixes will be implemented using a GitLab pipeline or GitHub actions to automate the deploy process when an update is released. In addition, updating the game to support new hardware and software (such as a new browser) will also be an important part of the maintenance process. This may involve updating dependencies to ensure compatibility with new operating systems or hardware. Regular testing will be performed to ensure that any changes made do not break the game, and unit tests will be run after each code push to ensure continued stability. Overall, the maintenance of the game will be an ongoing process, and all steps will be taken to ensure that it runs smoothly.

Conclusion

In conclusion, this software development plan has covered the main aspects of creating, deploying, and maintaining a game called "Space Race" using Pygame. The requirements for the game were described in detail, including the game objective, gameplay mechanics, graphical user interface, and hardware and software requirements. The design of the game included the game loop, sprite classes, and data structures used to represent game objects. The implementation of the game was explained, including the plan for development of each component and testing procedures. Deployment procedures, such as creating an executable and packaging the game, were discussed, as well as potential issues and solutions related to compatibility with different operating systems. Finally, the maintenance of the game was covered, with procedures described for fixing bugs, updating the game, and adding new features to ensure it runs smoothly over time. Overall, this plan provides a solid framework for creating a game using Pygame and ensuring its continued success.