Application Report (Invadem)

490424010

Project Description

Invadem is a 2D game developed in Java using the Processing library, similar to the popular arcade game Space Invaders. Players are tasked with the goal of moving and shooting from a tank to destroy a swarm of ever-approaching Invaders with varying traits. There is also a Two Player mode, which allows you to team up with a friend on the same keyboard to fight off the invaders together.

Iteration 1

Prior to the milestone, the App had much of its core functionality implemented. This included having a tank which could move left and right with the arrow keys and shoot projectiles at an approaching swarm of invaders. However, the structure of the App itself was not following many OOP idioms, as there were cases duplicate code between classes which could easily be inherited from common classes. This was mostly due to the limited scope of the project at the time, as it was hard to see which objects would require inheritance and which would not, without further knowledge of the additional requirements.

Iteration 2

Once the additional requirements were released, the entire project went through major changes even before any new features were added, to ensure the code would be highly modular and well structured.

One major change was creating an inheritance hierarchy for all objects found in the game. Before the additional requirements, Invaders, Barriers, Projectiles and Tanks were all independently implementing similar features (such as their position and size in space). With the introduction of new types of invaders and projectiles in the additional requirements, I decided to create an abstract class called *GameObject*, which contained an implementation of these variables and their associated *get* and *set* methods. I also created a child abstract class called *Entity*, which inherits these properties from *GameObject* and also contains more properties common to these objects such a state of *health* and the ability to be *destroyed*.
All these changes helped significantly modularise the code and also simplified the development of the additional requirements as I could simply extend upon the inheritance tree rather than reimplementing each feature separately.

Another major change was using an object-oriented state-based approach for handling different game modes as discussed in the lectures. This was because I found myself switching between running large blocks of code in the main App class using long else if statements, depending on whether the user was playing the Invadem game, or viewing one of the Next Level or Game

Over screens. With the introduction of a two player mode and main menu to satisfy the extension, I would require two more of these modes each with their own internal states. This would significantly increase the size of App and also require various distinct internal states to unnecessarily share scope. Thus, final implementation involved an abstract GameState class through which all five different game modes inherited, helping to increase modularity, decrease code complexity, and also introduced me to a new way of handling programming states which I had not yet encountered.

Final Implementation

App - *Class*
The App class extends from processing's PApplet, representing an instance of the entire Invadem game. This class is in charge of initialising the application and loading of all resources. It also holds various application-wide variables such as the Game State, High Score and Current Score.

Drawable - *Interface*
The Drawable interface allows objects which can be drawn to the screen to be clearly distinct from those which don't, and thus ensures the implementation of the relevant draw method. Drawable also provides objects with an optional static *loadResources* method, which is empty by default, as some drawable objects may not require the loading of any resources.

Collidable - *Interface*
The Collidable interface ensures all implementing objects possess methods to get position and size, to *hit*/*destroy* the object, and to determine if it has been destroyed. The reason for this interface was to allow some GameObjects to collide with others, while others don't, without needing to entirely split the inheritance hierarchy tree. An *isColliding* method is provided using the algorithm given in the assignment specifications.

GameObject - *Abstract Class*
This class is at the top of the inheritance hierarchy for the game, providing any objects which extend with both a position and size on the screen. This class also provides the various relevant getting and setting methods for such properties. As all GameObjects possess a position and size on the screen, this class also implements the relevant Drawable interface.

Entity - *Abstract Class*
This abstract class extends from GameObject, adding properties such as health, a destruction state, and the ability to be hit. The reason this class was incorporated was because some objects which require a position and size on the screen, did not necessarily need to interact with other 'Destroyable' objects (such as Button).

Tank - *Class*

The Tank class extends from Entity and is controlled by the player using key inputs. It can move left and right and also shoot Projectile objects upwards towards Invaders or Barriers.

Barrier - *Class*
The Barrier class extends from Entity and is a destroyable non-moving object placed in three groups of seven at the start of each level. Barriers can take three projectile hits and until they are destroyed.

Invader - *Class*
The Invader class represents a single entity within the swarm of invaders as they approach the player from the top of the screen, shooting Projectiles downwards at a set interval. There are also PowerInvaders and ArmouredInvaders which extend this class, inheriting all the same features but differing in that they deal more damage and have more health respectively.

Projectile - *Class*
The projectile class represents a small object which will deal damage upon collision with all other Entities. PowerProjectiles extends this class, which has all the same properties except it will instantly destroy any object it collides with and is also slightly larger.

GameState - *Abstract Class*
The GameState class is an abstract representation of one of the 5 possible states the game can exist in: Menu, OnePlayer, TwoPlayer, NextLevel and GameOver. Each state independently handles it own user input and implements its own draw method. The reason for this class was to significantly reduce the amount of code within App, and also split the entire game up into multiple logical sections.

Reflection

One potential improvement I could make to the application would be to better utilise the existing Interfaces, as they are currently only being utilised by a single class each, making their existence outside of the implementing classes somewhat irrelevant. I had initially created them as I thought there would be certain functionalities which could not easily be distributed through class hierarchies, however due to the limited scope of the application they were needed far less than I had anticipated. If, for example, the scope of Invadem was to expand to include some GameObjects which were not Drawable, the use of these interfaces would be justified.

Extension

My extension to the design specifications was to implement a two player game mode, allowing two distinct tanks to be controlled using different keys on the same keyboard. This extension also required the addition of a Main Menu to allow the user to select between One or Two player modes.