Project Report

Project Description

Student ID: 480048691

This document is a game design report for the game Invadem. This project is a single (two) player game which involves the player(s) controlling a tank to destroy the invaders before they reach the barriers or the tank is destroyed. Every 5 seconds, an invader will be chosen randomly to fire a projectile at the tank. The game is won by destroying all invaders (40). Levelling up increases the fire rate of the invaders. A single level game can take up to approximately one minute.

Iteration 1

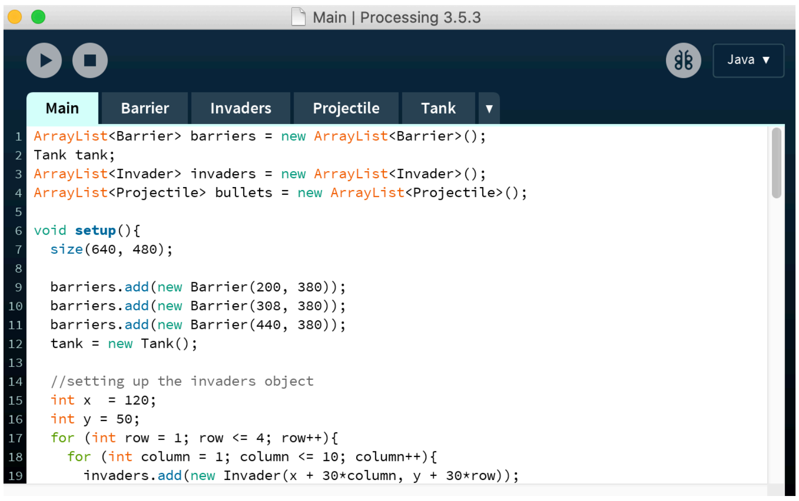
Designing Game Loop

1. Identifying main entities for the program
   - After reading the specification and interpreting the given additional resources (MovingSquares), I identified the main entities of the program including Tank, Invader, Projectile, Barrier with common instance variables such as x-coordinate, y-coordinate, image, and speed.

   ⇨ Created an abstract class Entity and child classes Tank, Invader, Projectile, and Barriers.

2. Drawing the objects onto the sketch
   - Having trouble understanding Papplet and associating classes with the main class

   ⇨ Drew each entity onto a sketch using the Processing program itself



```
ArrayList<Barrier> barriers = new ArrayList<Barrier>();
Tank tank;
ArrayList<Invader> invaders = new ArrayList<Invader>();
ArrayList<Projectile> bullets = new ArrayList<Projectile>();

void setup(){
  size(640, 480);

  barriers.add(new Barrier(200, 380));
  barriers.add(new Barrier(308, 380));
  barriers.add(new Barrier(440, 380));
  tank = new Tank();

  //setting up the invaders object
  int x  = 120;
  int y = 50;
  for (int row = 1; row <= 4; row++){
    for (int column = 1; column <= 10; column++){
      invaders.add(new Invader(x + 30*column, y + 30*row));
```

3. Method determining the movement of the invaders
   - Set moment state 1 (left to right), 2 (downwards), and 3 (left to right) to enable continuous movements.
   - Set two new instance variables "checkpoints" called xPos and yPos to determining when to change the invader's movement state

4. Mechanism for tank to shoot projectile
   - Add the start, I created two seperate ArrayLists to distinguish Projectiles object coming from an Invader or the Tank.
   - Every time the space bar is pressed, a new Projectile will be instantiated at the tank position, and made to be "displayed".
   - The same mechanism was applied for Invaders.

5. Mechanism for detecting collision
   - This method was implemented in the App class and contained the algorithm provided from the spec with the parameters (Entity e1, Entity e2).
   - To invoke this method, a for loop is used to loop over the bullet lists to detect collision between any Projectile bullet in the list with another object (Tank or Barrier).
   - At the time of milestone submission, this method was working but was very inconsistent.

6. Setting winning and losing conditions
   - A global variable gameState was implemented to enable transition within the game, whose default value is 1 for main game loop.

   Losing condition (gameState = 2):
   - A global variable barrierLimit whose value was assigned during the process of setting up the Barrier objects was used to determining the first losing condition of the game.
   - The tank object has its own 'hit' variable which was incremented every time the checkCollision method returns true for Tank and a Projectile object. When hit = 3, gameState will change to 2.

   Winning condition (gameState = 3):
   - A global variable score was created to keep track of how many invaders the player have destroyed. When score = 4, gameState will change to 3.

Iteration 2

1. Moved collision detection method to the Entity class
   - After milestone submission, one of the suggestions came from the tutors for my code was to move the collision detection method to the Entity class. I also changed the algorithm to my own method. After doing this, the program ran much faster and is now consistent.

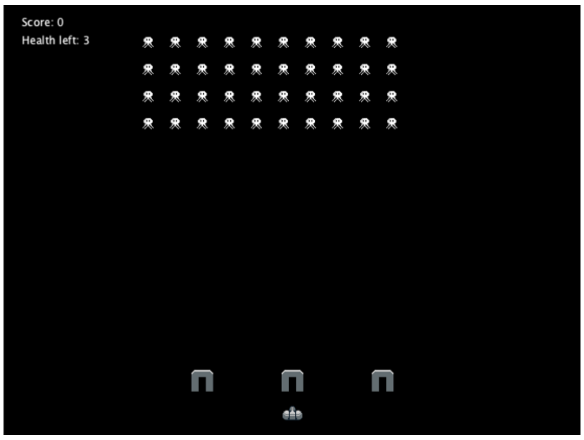
2. Armoured, Power and Regular invaders
   - Create new classes for Armoured, Power, and Regular invaders extends from the super class Invader to implement new properties such as Health and Score Value.

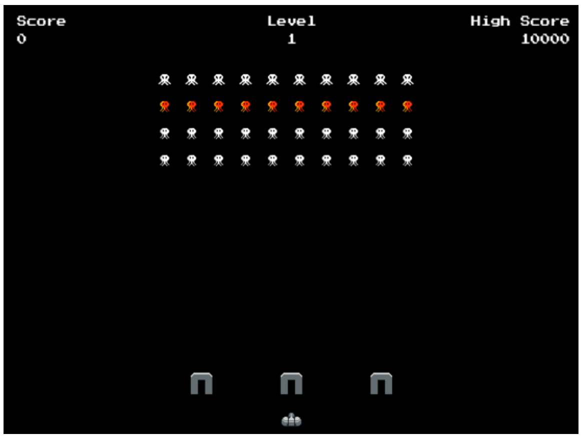
3. Changed the method for determining the winning state
   - Instead of checking for Score = 40, since the score value is now changed, I changed this method to check for the size of Invader list to be 0.

4. Two different types of Projectile
   - There are certain information need to be keep record of a projectile such as which shooter it is coming from (from a tank or an invader) and also the level of damage, these two new variables were added to the Projectile class.

5. Moved the corresponding methods to its respective class
   - Methods such as fire for both methods Tank and Invader were moved to their respective class.
   - The Tank, Invader, and Barrier classes each have a method called interactionWithBullets that will loop over the list of projectiles and check for collision between the respective object and a projectile. If collision detected, the projectile will be set to be removed and will be removed using another function. The respective object will also call its gotShot() method to determine the level of health it has left. If the object's health <= 0, the object will set to be removed.
   - Another function called removeDeadInvader() (or removeDeadProjectile()) will be called right after the for loop above to loop through the invaders (or projectile) list to remove any invader that has been set to be removed.

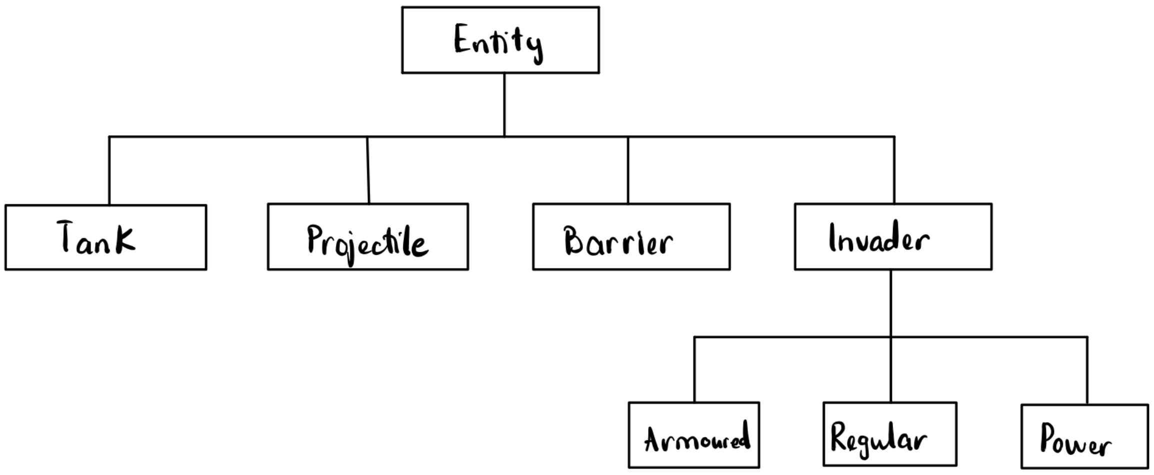
6. Change in the display of the main game loop





Final Implementation

1. Entity class and its subclass



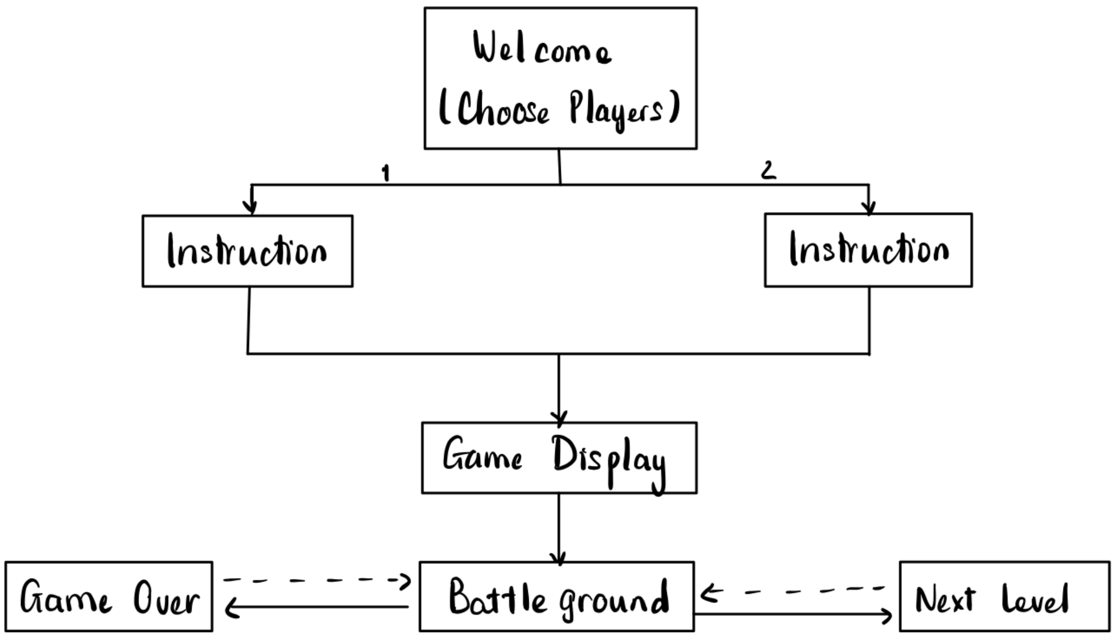
- Each classes of the following (Tank, Projectile, Barrier, and Invader) were created to keep track of an object's location on the sketch, its speed, health, and associate methods such as movements, display, and other specialised methods such as fire, interactions with bullets, changing sprites and removing the object if applicable.

2. App class
   - The App class starts off with the setup() method, which calls the restart() method.
   - The restart method will set up the objects to its initial states (which helps with gameState transitions later on), note that the objects to not yet drawn.
   - After that, the gameDisplay() method will be called which will call the methods displayTank(), displayBullets(), etc. Each display function will call the following instance methods: draw objects, object movements, detecting its collision with other objects, and remove object if needed.
   - The GameWon() and GameLost() method will be called if the condition(s) is met.
   - Other methods relating user interactions with the game such as information display, nextLevel/gameOver screen, scoreIncrement, etc. are also in the App class.

Reflection

- There are a few methods that I believe can be created using only another abstract class inherits from entity whose subclasses can be Tank, Invaders, and Barriers because they all interact with Projectiles. Furthermore, I also realised having the Barrier with speed = 0 can be of second-rate.
- After this project, I understood more what it meant by "Object-Oriented Programming" and how I should have methods that only involves a single object to be within its own class.

Extension

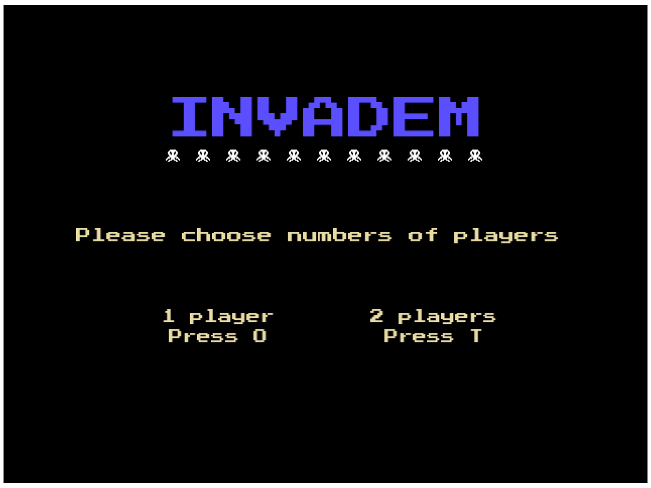


- I created two different game modes, single player and two players. A welcome menu appears once the program is ran. Player(s) can choose the number of players by pressing either O (one player), or T (two players). After the number of players is chosen, an instruction menu will be displayed showing the keyboard commands for each player. Player(s) can also press backspace to go back to the welcome menu. To enter the game, player(s) press Enter.

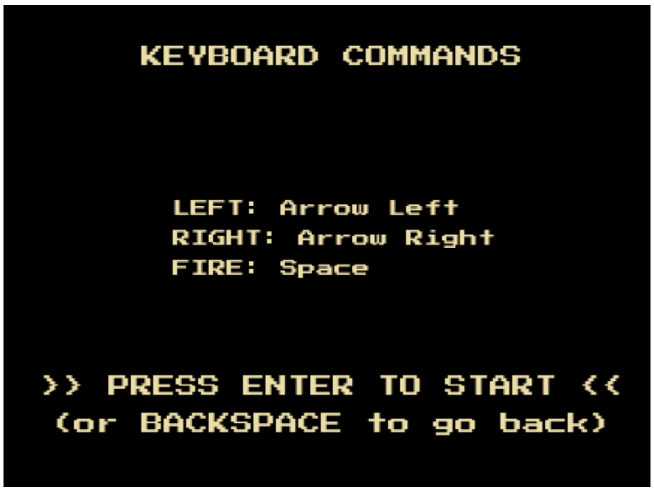
Two-player mode:

- Cooperative, i.e. players play together to remove the invaders. Hence, the current score shown will be the score gained by both players. Game is lost when either of the tanks is destroyed or the invaders reach within 10 pixels of the top barrier.

Welcome Menu

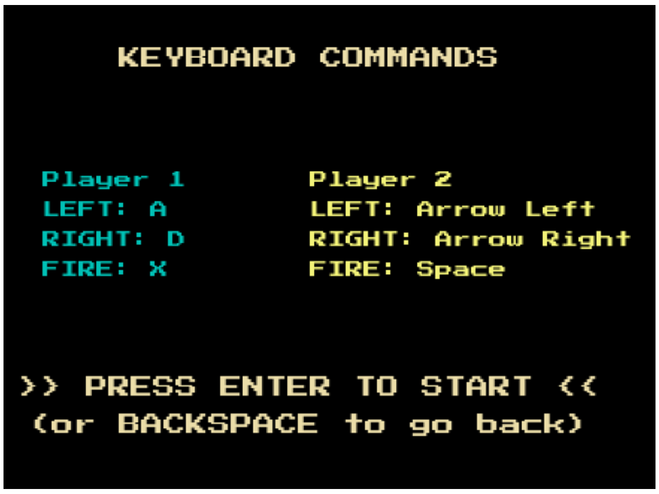


Instruction Menu


KEYBOARD COMMANDS

LEFT: Arrow Left
RIGHT: Arrow Right
FIRE: Space

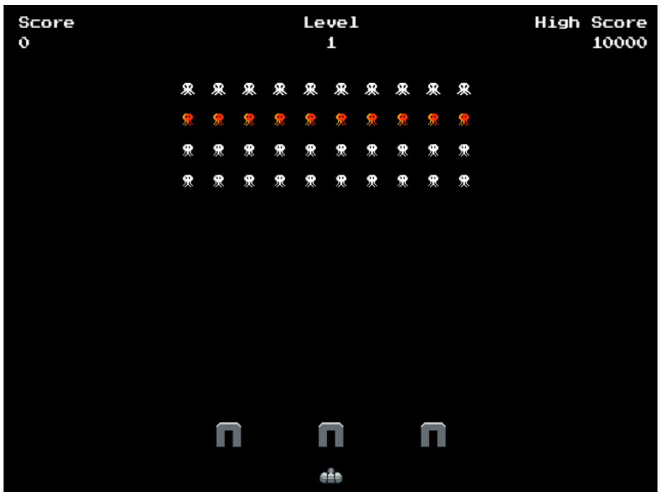
>> PRESS ENTER TO START <<
(or BACKSPACE to go back)

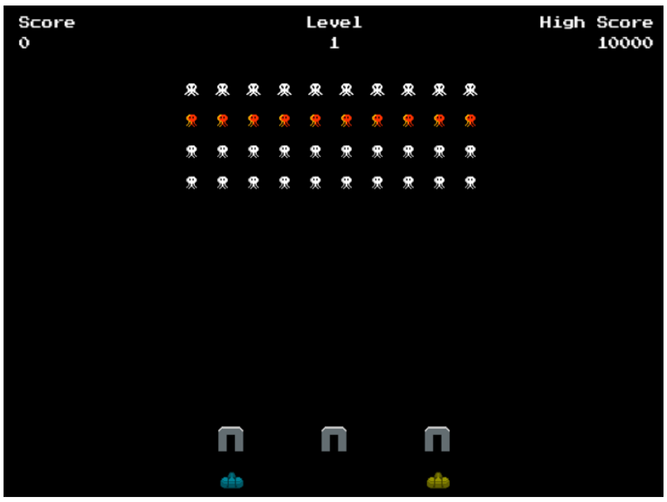

KEYBOARD COMMANDS

Player 1	Player 2
LEFT: A	LEFT: Arrow Left
RIGHT: D	RIGHT: Arrow Right
FIRE: X	FIRE: Space

>> PRESS ENTER TO START <<
(or BACKSPACE to go back)

Game Display


Score
0
Level
1
High Score
10000


Score
0
Level
1
High Score
10000