CS4303 W03 Practical
150015752
3rd October 2018

Overview

The task of this practical is to design and implement an artillery video game which features 2 tanks who try shoot each other at turns while dealing with hills in the environment and wind changes. The game should provide option to play in 2 players or against AI, it should have a user interface, including wind information and tank force and angle information. Furthermore, the game should include collision detection so that tanks cannot go through walls and each other. Shot projectile should be subject to gravity, wind forces and the starting acceleration.

Execution

ProcessingStart.jar file is in *out/artifacts/ProcessingStart_jar* folder and can be run by the command:

```
java -jar ProcessingStart.jar
```

In the case of wrong version, reset version in intellij and build to jar again.

Design and Implementation

Game Object

Each game object in the game inherits from the base *GameObject* class. Game objects implement Component or Strategy design pattern which means, that each beahviour of a game object, such as input or collision is stored in a separate class and referenced in the game object as a field. There are certain shared variables, such as position or velocity which can be accessed by any component but the main goal is to decouple the behaviour from the object and deal with each one separately. This means, that with right interfaces, a component can be changed without any change to the base object. For example, we can change box collider for a circle collider and no change in the *GameObject* class is required. The disadvantages is that with decoupled behaviour there is an additional step to go through which can have negative impact on the performance.

Components

Components include colliders and input beheaviours.

For colliders, we can decide between circle and box colliders which are initialized by slightly different parameters. Colliders than detect collision with other objects and change the velocity of the game object accordingly. When touching, however, there’s a certain limit to velocity of an game object, after which the object goes through another object because the collision wasn’t even detected.

Inputs include 2 options: player or ai. In the case of player input, angle is set based on angle from tank to the mouse and the tank head moves accordingly. When loading, an arrow appears, and points in the direction of shooting. It lengthens with higher force and stops at maximum force of 20. AI input is similar but uses different methods to calculate angle and force. To acquire angle, the AI looks at the highest block on the map and sets the angle so as to overshoot it. Then it only changes force based on last hit position of shell.

Game manager

Game manager actually manages the game enviroment and rules of the game. It determines winning situation, resets the round, changes game UI based on player input and score of the tanks and many other tasks. It also generates the environment of hill randomly and resets it when a game is won. As there is only one Game manager per game, we can use Singleton design pattern on it. The Game manager class also inherits from the *Papplet* class, and therefore is the starting point of the application. Game manager is storing point of all the behaviour governing the game. The other alternative would be to store it in the game objects themselves, which would be impractical and too robust.

The rules of the game are, that it requires only one hit kill an opponent and it takes 3 kills to win the game. At the start of every game, tanks are falling from a great height and can be moved which is an intention. Players can find ideal positions for shooting. The wind changes every third move, so that will become more adaptable to differenet wind factors over time.

UI

The UI include start menu with options to play against another player, play against AI and exit. The buttons are highlighted when mouse is over them. When playing, a score is displayed in the upper corners of the screen and the wind arrow is displayed in the left upper corner, under the score of the left player. Furthermore, when a round or a game is won, an appropriate message is displayed. Each tank’s shooting information is also displayed above them and moves with their position. The UI functions as a component in that it can changed for another with the same interface, however, it is not referenced in a game object but rather in Game manager as it depends on many rules of the game and Game manager is where such behaviour is stored and so easily accessed.

Physics

The objects have a velocity which is stored as a vector and therefore displays both power and direction of a force. These vectors can then be added or subtracted to from each other and we have an easy way to use forces that affect an game object. Any object can be subject to gravity. The gravity is implemented as a constant acceleration of velocity in the direction to the ground. To slow this acceleration,we use a drag, which mulitplies any resulting velocity by a constant between 0 and 1.

When firing a shell, a starting acceleration is added to the shell’s velocity, and the velocity is then affected by wind vector, gravity vector and the drag, which all afflict the velocity every frame. Each block has a collider and is also subject to gravity, therefore if we shoot a block under any other blocks, the ones above it will fall down.

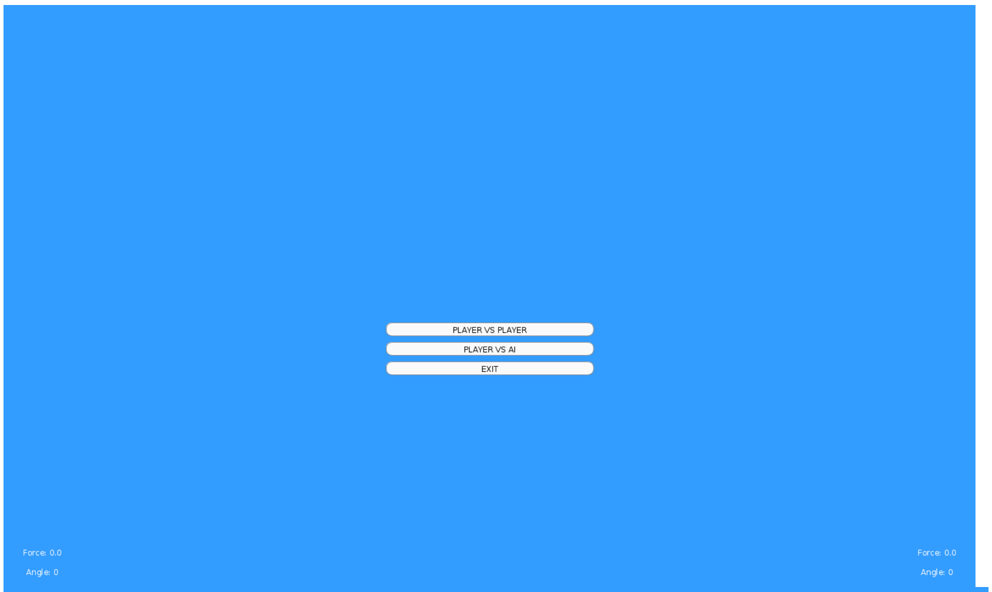
Conclusion

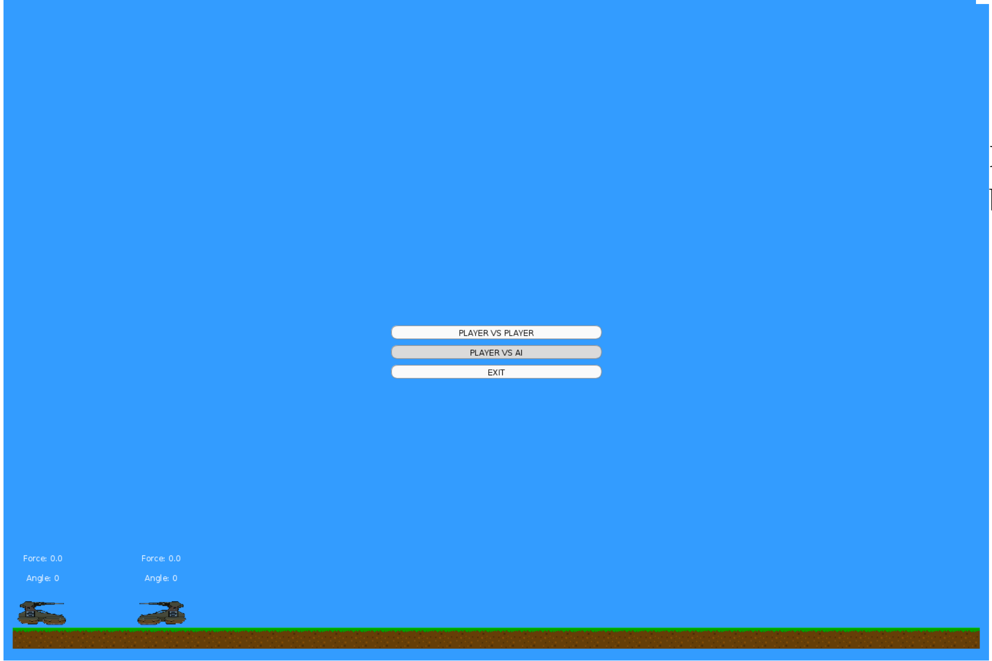
I implemented a functioning game with the base rules specified in the assignment. The game objects use components to decouple the behavious from each other and the game object themselves, while the behaviour of the game is governed by Game manager. UI displays all the relevant info. Colliders work as expected, if a velocity is not too high. Player can choose to play against another player or a simple AI which shoots with improving accuracy over round. An extension include blocks with colliders which are affected by gravity, and can therefore fall if not supported by a lower block. The most difficult part of the practical was to implement the game object behaviour, the Component design pattern and, especially colliders. Given more time, I'd definitely improve colliders and do more testing.

Testing

The testing includes Junit4 tests in the Intellij project provided and images provided below/

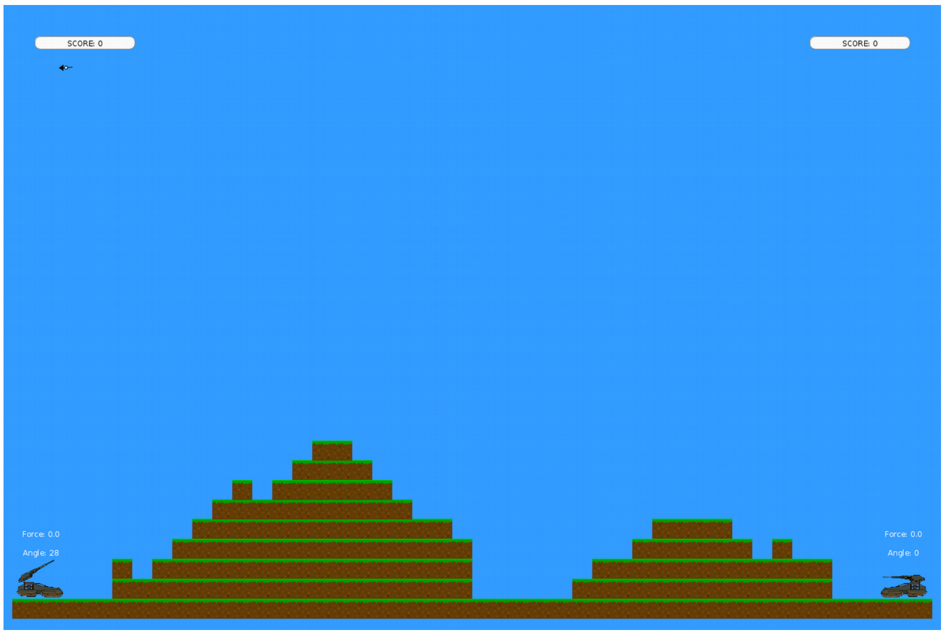
Displaying the start menu

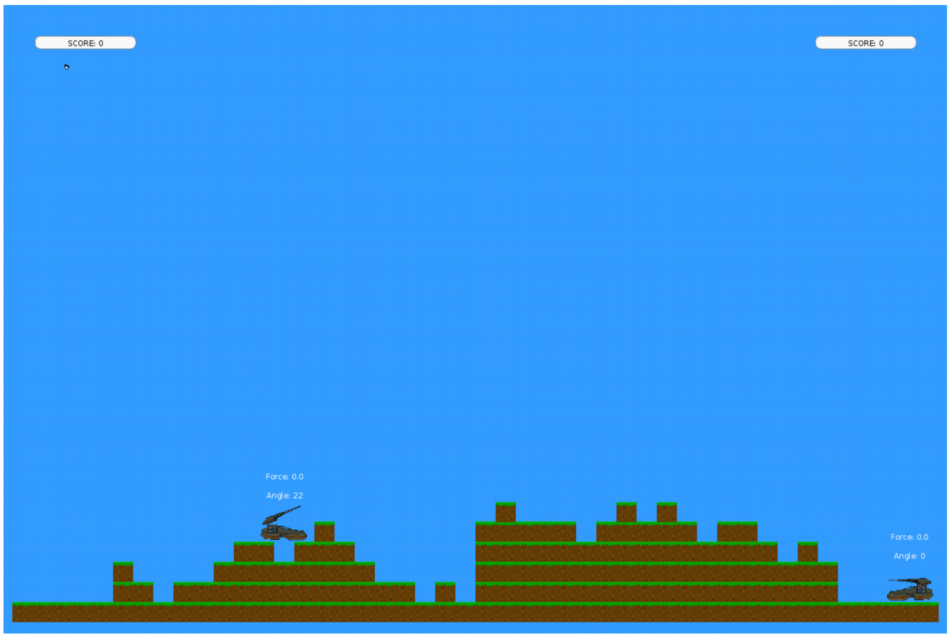




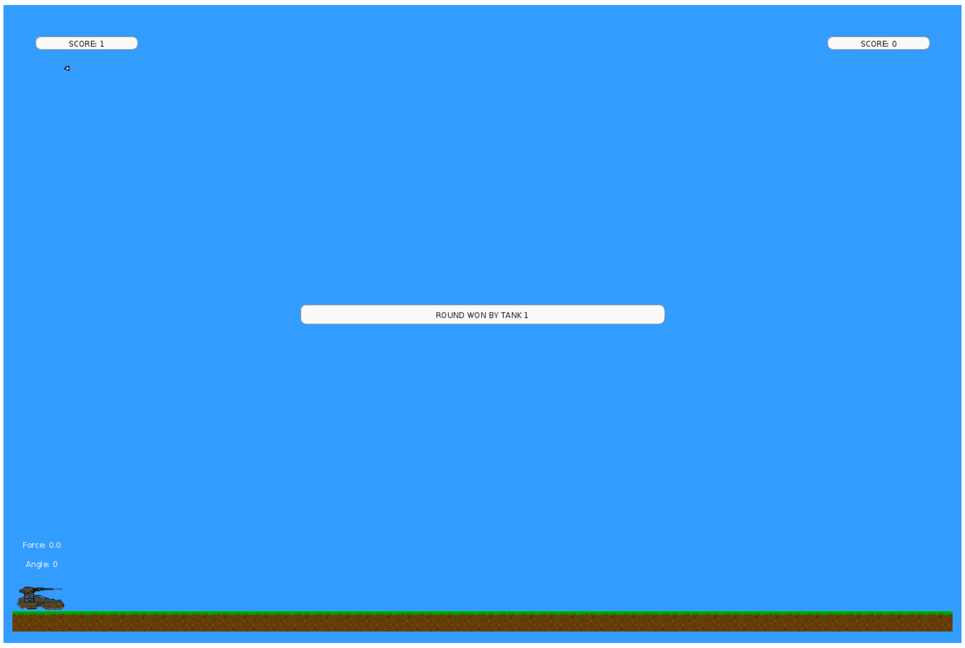
Highlighting the buttons

Random generation of the hills

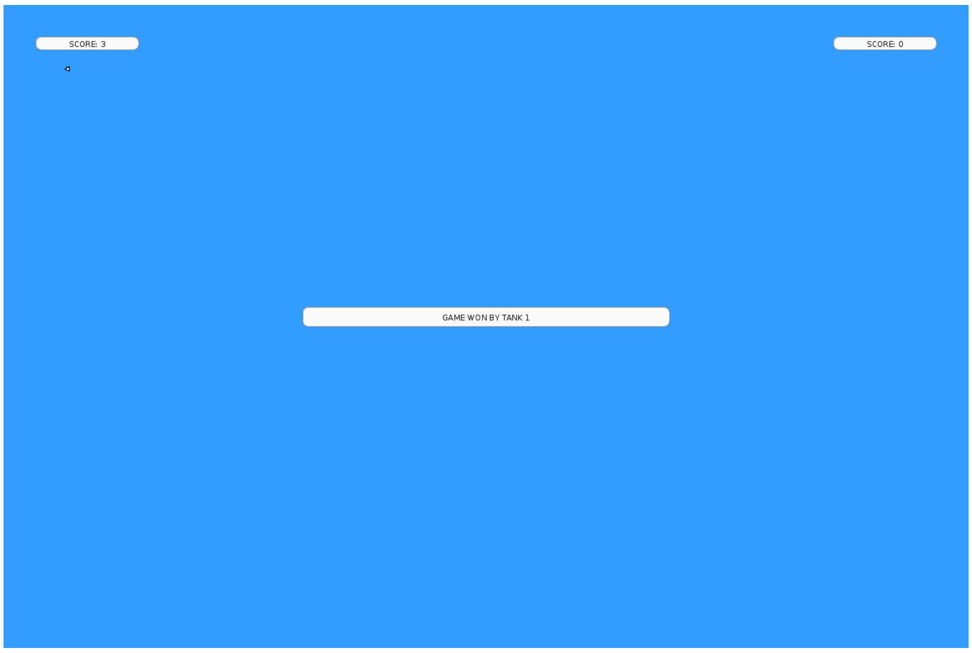




Round won

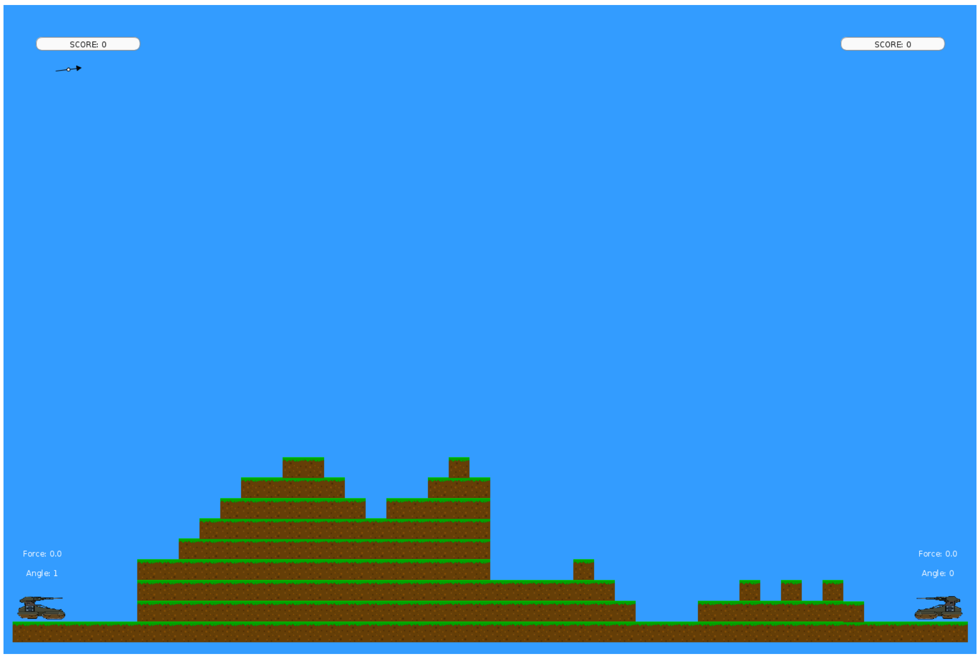


Game won

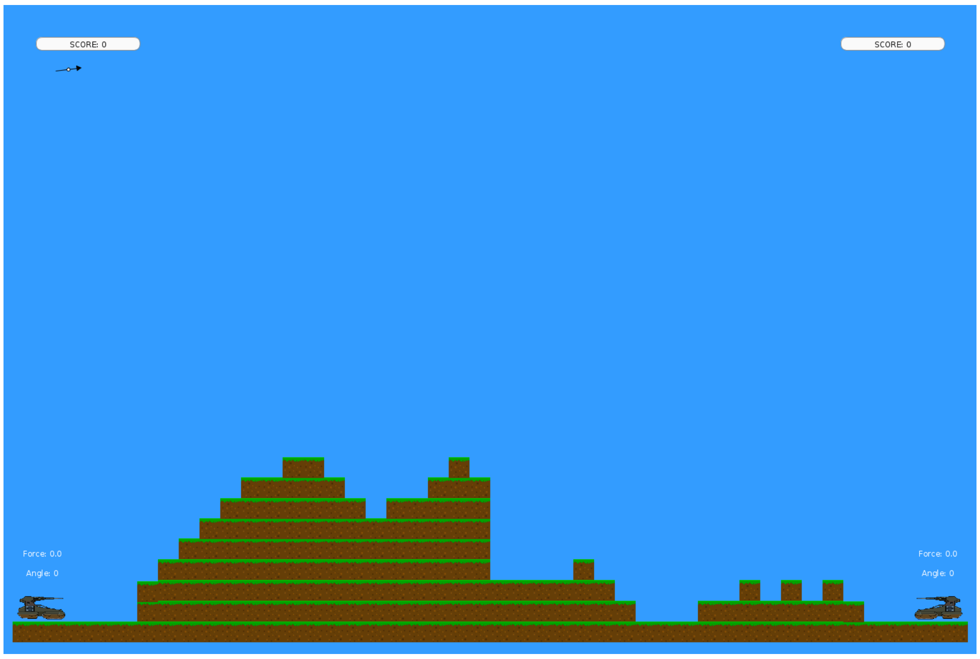


Displaying block destruction and block gravity

Before



After (the leftmost lowermost block had been destroyed and the upper ones fell)


SCORE: 0
SCORE: 0
Force: 0.0
Angle: 0
Force: 0.0
Angle: 0