Bulldog

Java android game

Module code:	CS2JA16
Assignment report title:	Android game
Student number:	27005943
Date completed:	16/03/2020
Actual hours spent for the assignment:	Approx. 40
Assignment evaluation:	

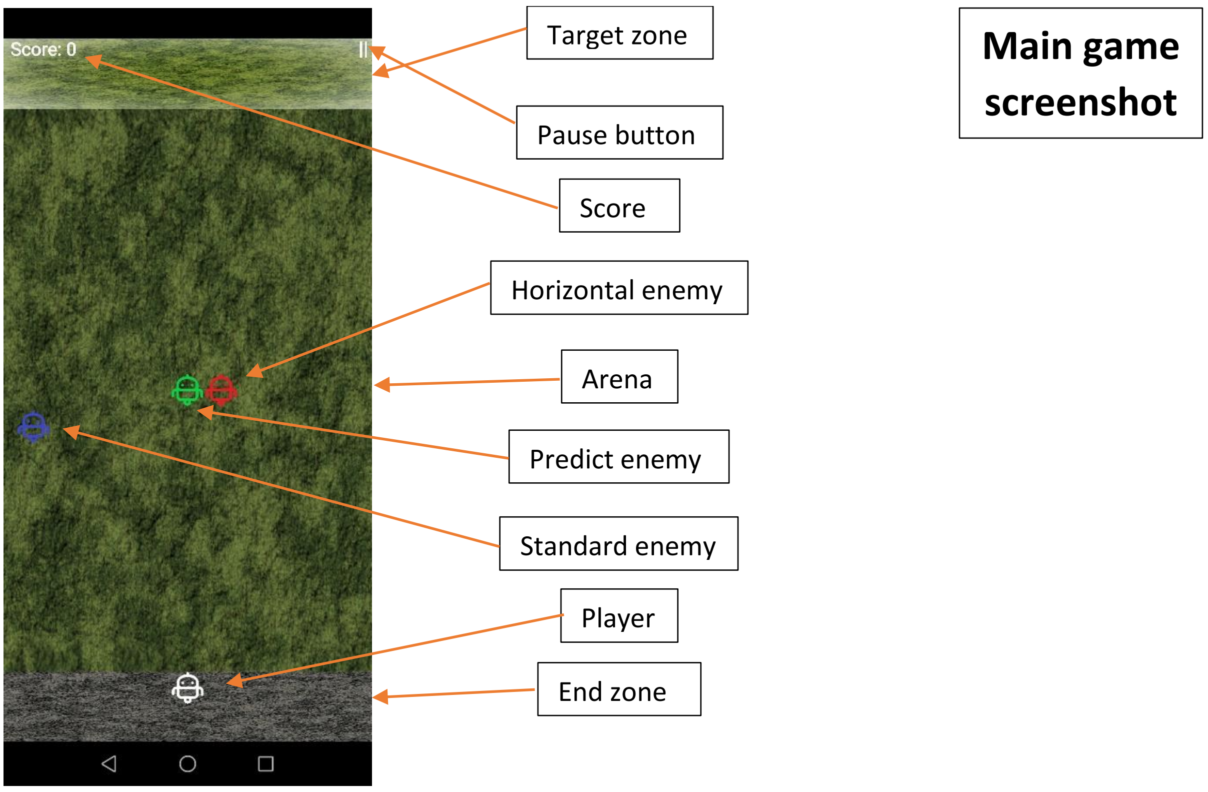
Abstract

I have been given the task to create an android game. The game itself needs to be complex enough to engage the user but simple enough to enable the user ease of play. Creating this game has shown the sheer amount of time it takes to make a game look pleasing as well as render quickly.

Introduction and showcase

For my android game, I have decided to base it on the real-world game bulldog. The player will be required to make it from the bottom of the screen to the top of the screen, a set number of times, without getting caught by the enemies. Both the player and enemies will move on around on a 2D arena.

There will be a small safe zone at the top and bottom of the screen where the player can stay without the enemies chasing them. However, once the player leaves the enemies will immediately continue their pursuit.



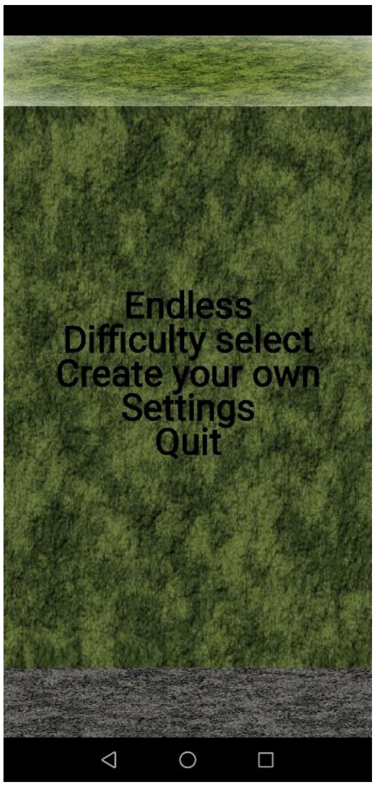
Enemies will be able to move in one of three ways:

- Standard – This enemy will move directly towards the enemy in a straight line. When the player is in a safe zone, this bot will move randomly until the player leaves.
- Predict – This enemy will calculate where the player will be in 1 second time and then move towards that location. If the player is in the safe zone this bot will move to the centre of the arena.
- Horizontal – This enemy will move horizontal across the screen until its *x* position is very close to the players at which point it will move up (or down) towards the bot. The player being hidden will mean this bot will move to make its *x* position central.

Upon loading the game, the first screen displayed will be the main menu. The main menu will have 5 different options:

- Endless mode

- Difficulty select
- Create your own
- Settings
- Quit



Main menu

The *endless* mode will allow the player to continuously play trying to reach a high a score as possible. The game, in this mode, can only end by the player being caught by the enemy or exiting back to the main menu from the pause menu.

In *difficulty select*, the player will be able to select a difficulty from easy to hard. This will produce a level for the player to complete. Unlike the endless mode, the player will have an end goal. For all difficulties, the end goal is to go back and forth from the top of the screen to the bottom as set number of times without getting caught. If the player succeeds, a win message will appear. However, upon getting caught the level resets.

Create your own will allow the user to create their own level. The player will get to pick the number of enemies and what different types the enemies are. Then it will continue like endless mode. There is a limit on the number of enemies.

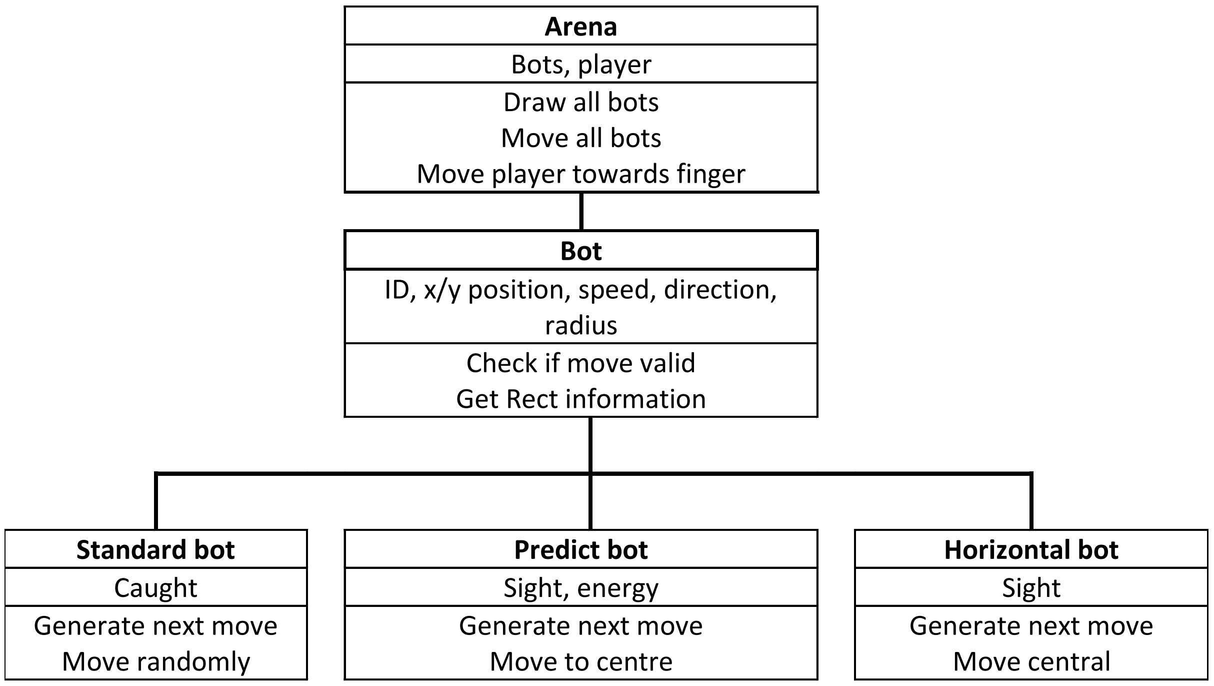
For *settings*, the player will be able to change the look of the game. This includes changing what the arena looks like and what the bots look like. The player is able to change what each one looks like because in the code I read the images as *.png* files.

Quit option allows the user to close the game down.

OOP design

When designing the game, I first started with the *ConsoleLevel* and made sure it was displaying all the bots correctly (even though the bots could not move).

I then created the arena for the bots to move around in. The included all the collisions and moving all the bots.
After this I was able to create the bots themselves. I used an abstract *bot* class and inherited it for all the diffract types of bots and the bot the player controlled.
Finally, I created the main menu page at the end. The main menu is just a series of button that take the user to different areas of the game itself.



Memory usage and speed improvements

For memory usage, I looked at the profiler built into android studio and saw that the most memory usage I use is 200MB which isn't that much. Therefore, I have decided not to change anything for the memory usage.

Originally my game was rather slow. That is because on every frame I was redrawing the entire background as well as all the bots on the screen. I therefore changed this to only update the background when it is needed and only update the bots when they move. Doing these changes means the game runs smoothly.

Improvements/extensions

One big improvement I would love to make on my app is the graphics. At the moment, the look of my game seems clumsy. If I had more time, I would have made it seem more professional looking.

In the game, I allowed the user to create their own levels. This allows the player to pick what enemies they face and how many they face improving their skills at the game.

Another thing I decided to add to my game was the ability for the user to change what the background and bots look like. If the user goes onto the settings options, they will be able to change the background and bots to fit their own desire rather than the programmers choice.

Demo self-assessment sheet

Each tick-box represents one mark unless otherwise specified.		Range
Code style (2 marks each points): Following Java code conventions for all the project [√] variable and class names [√] method names [√] indentation rules [√] Using inline comments frequency [-] Javadoc comments for every function (except getters/setters)	Overall OOP design and API usage: [√] Appropriate use of inheritance (2 marks) [√] Appropriate use of Abstract classes (2 marks) [√] Use of Android API classes/methods (Other than in base code) (3 marks) [√] Greater than 3 original classes (3 marks)	0-20
Functionality: [√] On-screen menus (1 mark) [√] Scores (1 mark) [√] Controllable character (1 mark) [√] Sensor interaction (touch/ accelerometer/ etc) (1 mark) Game has levels [√] Works (4 marks) [ ] Attempted (2 marks) Use of standardised data structures (e.g., List, Vector, Collection, Date): [√] Works (10 marks) [ ] Attempted (5 marks) Randomly/procedurally generated features: [√] Works (10 marks) [ ] Attempted (5 marks) Opponents (AI): [√] Works (5 marks) [ ] Attempted (3 marks)	Design quality (both game and other game screens) (1 mark each): [-] Professional looking [√] Understandable game flow [√] Feedback to user instead of crashing, or recover [√] Runs smoothly without interruptions [√] Installs without error [√] Starts/exits without error [√] Program does not crash [√] Produced in video form	0-40
Improvements marks: Online high score list [-] Works (10 marks) [-] Attempted (5 marks) Multithreading improvements [-] Works (10 marks) [-] Attempted (5 marks) Memory optimisation [√] Works (5 marks) [ ] Attempted (3 marks) User Level Creation [√] Works (5 marks) [ ] Attempted (3 marks)	Other extension/improvement / innovation [√] Works (10 marks) [ ] Attempted (5 marks) Note: either the attempted marks or the works marks will be given (so you can only receive 5 marks per attempt max, and there is a max of 10 marks for this section but tick off any extra work done anyway!) For attempted marks to be awarded the feature must be at such a state, that the code could have worked but does not due to bugs or similar.	0-40

Demo self-assessment examples:

- Appropriate use of inheritance - (Example of Class in your code that inherits other class

```
public class Player extends Bot {
```

- Appropriate use of Abstract classes - (Example of Class in your code that is an abstract class)

```
public abstract class Bot {
```

- Use of Android API classes/methods - (Example (one or two would fine) of some API classes/methods)

```
import java.util.ArrayList;
import java.util.Random;
```

- Use of standardised data structures (give some example belonging to one of these e.g., List, Vector, Collection, Date)

```
private ArrayList<Bot> Bots;
```

- Randomly/procedurally generated features (e.g., randomly adding some obstacles, some sceneries, players, etc)

```
x = Arena.randomNumber(safe_zone_height + bot_sizes, screen_size[0] -
safe_zone_height - bot_sizes);
y = Arena.randomNumber(safe_zone_height + bot_sizes, screen_size[1] -
safe_zone_height - bot_sizes);
```

- Example of AI Opponents (automatically acting objects, like automatically changing position, chasing other objects, opponent moving with the movement of player, etc.)

```
protected void chasePlayer(Arena a) {
    int[] wanted_pos = a.getPlayer().getPos();
    moveToPosition(a, wanted_pos);
}
```

- Example of Online Scoring

N/A

- Multithreading improvements

N/A

Conclusion

I was overall happy with the outcome of my game. If I had more time, I would improve the overall look of the game. The feel of how the player bot moves around and the other bots interact is really satisfying.