

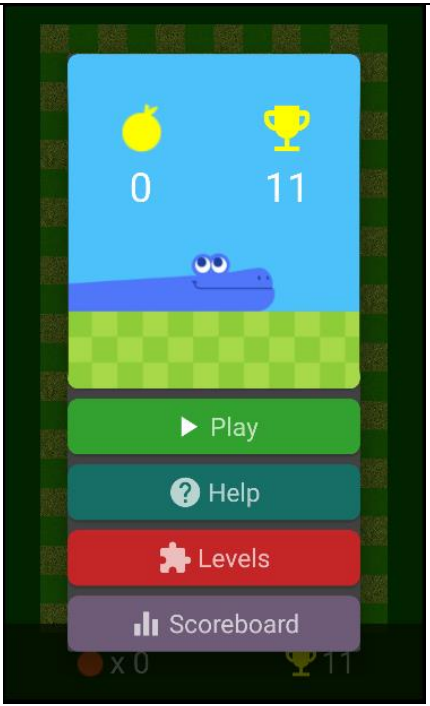
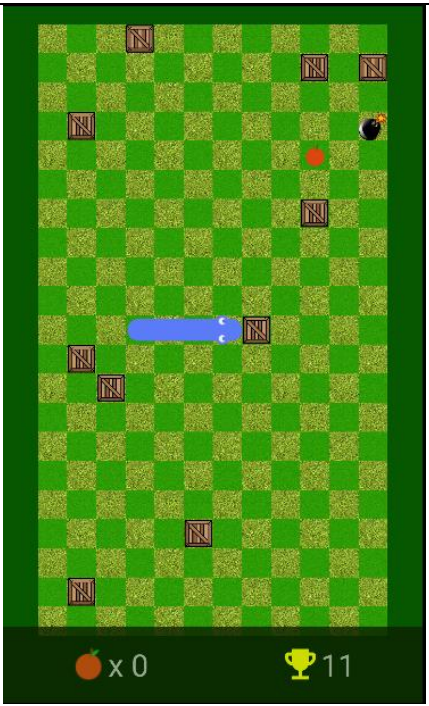
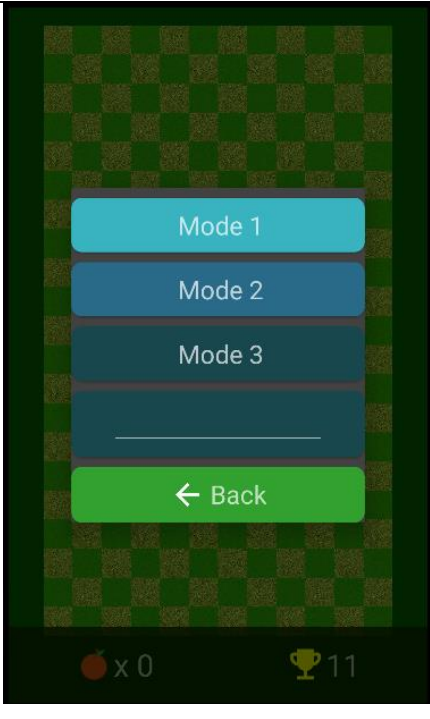
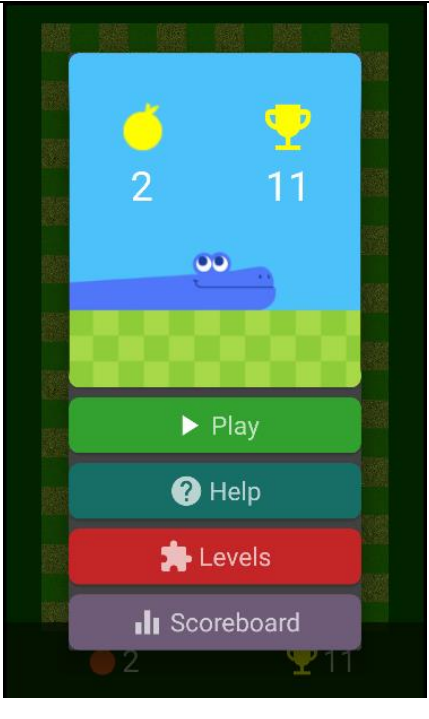
FRONT PAGE

- Module Code: CS2JA16
- Assignment report Title: Android Game
- Student Number: 29003617
- Date: 25/09/2022
- Actual hrs spent on the assignment: 25
- Assignment evaluation (3 key points):
 1. Was not enough lecture material for multithreading.
 2. Not enough material/explanation on memory and speed optimisation.
 3. The practical lessons were not long enough to get sufficient help.

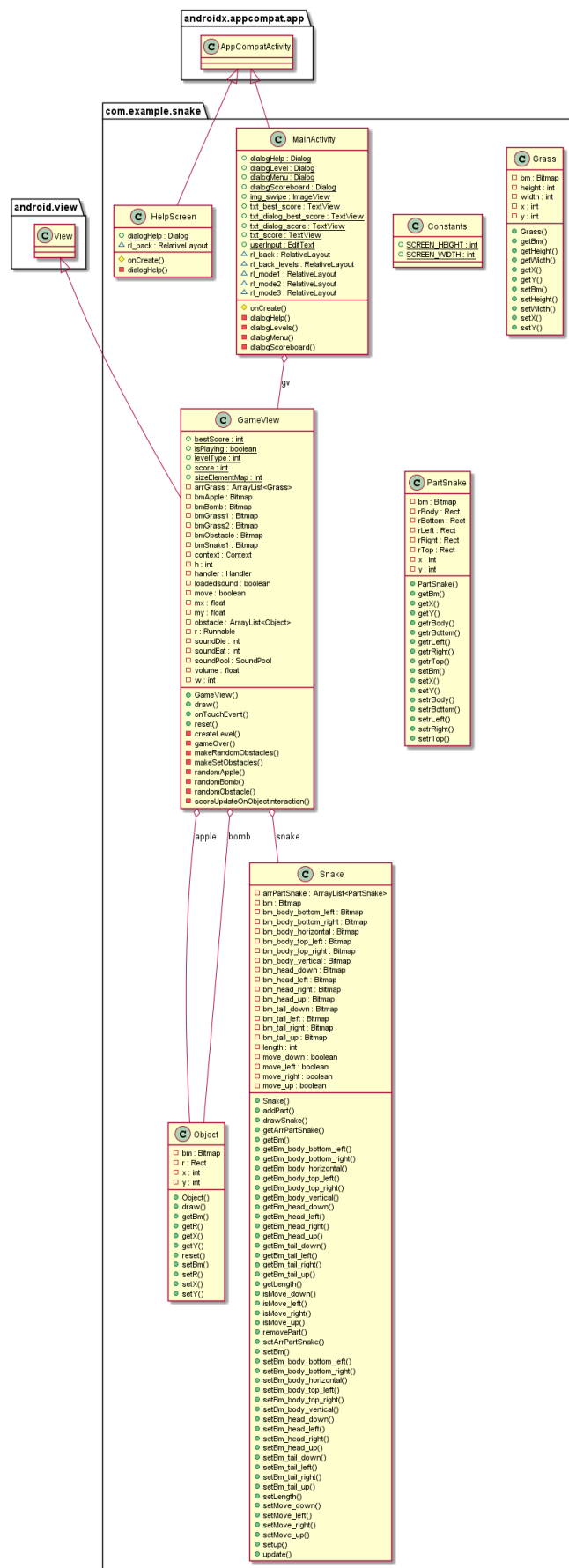
Snake

Based off a retro game called "Snake" this is a simple single player game where the player competes to collect points and competitively beat a previous score. There are several modes for the player to choose from thus allowing the player to keep challenging themselves. This project introduces programming techniques and data structures which aid on tackling problems throughout the code.

Introduction and showcase

	<h3>Menu</h3> <p>The first screen the user will be presented with is the menu screen. This consists of several buttons which will direct the user to another screen, these hold specific functionalities and purposes.</p>		<h3>Game/ Objects/ Snake</h3> <p>This image illustrates the objects which are drawn into the game, these are background, enemies, obstacles, apple, and the snake. They all have their unique features included into the game. Each entity is represented with an image, this clearly differentiates the characteristics of each object.</p>
	<h3>Levels/ Input</h3> <p>This is the level screen this allows the user to choose a certain mode. Each mode from top to bottom increase by difficulty, this allows the player to evaluate their skills as well as making the game less redundant. The player is also able to add a random number of obstacles throughout the game with the text field.</p>		<h3>Scoring/ Scoreboard</h3> <p>After each iteration of the game the player score will be tracked, after obtaining an apple the player will increase their score, their total score is shown under the trophy symbol, and their current score is shown under the apple. The best score is then added into the scoreboard, this can view at a later date.</p>

SNAKE's Class Diagram



OOP Design

The main activity holds all the relative information about running the game as well as starting the relative menus. The main activity calls the class “Game View” this gets declared and used when the play button is pressed. The main activity also initialises all the buttons presses and scores the user will interact with; this is needed as these are variables/objects which will be used several times through the app process. The help screen and main activity screen both use intent, this shows there are multiple screens in the app.

The “Game View” holds all the information about the game this includes the map, apples, points, snake, obstacles. These are necessary data for the game to function as each entity have unique functions, they are separated with their own classes holding unique methods and attributes. This introduces modularity to the development of the game; this also helps when tackling each object efficiently.

Inheritance will be introduced with a generalised “Entity” class this will include common attributes and methods which all inherited/child classes share. The child classes in this case would be “Snake,” “Part Snake,” “Grass,” and “Object” these are all entities which are drawn onto the game thus these share variables which are repeated, inheritance reduces code and introduces a simpler code base to organise and understand.

“Snake” and “Part Snake” coexist together as “snake” holds an array list with the data type “Part Snake,” as each section of the snake’s bitmap is stored into the array list. The snake bitmap holds images which are split into segments used to display each body part of the snake’s structure, therefore several variables are declared. These variables are private, this shows encapsulation is considered thus not allowing other classes to obtain/access the data in the “Snake” class. This can also be seen in the classes for the other objects in the game.

Polymorphism was used to draw objects into the game as there were several scenarios for entities to be drawn onto the game, such as the user being able to choose how many and where objects should be displayed. The method “makeRandomObstacles” and “makeSetObstacles” both hold the same code however the difference being the parameters sent to the method. The error here is the names of the class as this does not show true polymorphism.

Memory usage and Speed improvements

As the objects in the game are constantly being created, they also need to be removed otherwise the system will be overloaded this will cause overloading of system resources this will in turn cause dysfunctionalities while the code is still running. This error was established when the array list holding all the objects for the map as well as the entities on the screen were not cleared/emptied when the player chose to replay the game, this caused the framerate to severely be overwhelmed and lagged the game. The game was functioning too slowly as the resources were being overloaded by loading too many objects onto the screen, therefore the user interaction and experience was uncomfortable. To tackle this problem is to clear the array list and reset the entities when the user replays the game, this removes all previous objects which were drawn onto the canvas therefore freeing any unwanted space on memory thus the speed of the game was not hindered.

Another method to improve memory management is by using global variables this can be seen as a separate class is created to store "Constants," such as the system screen x and y values. This can be further increased as more final static values which are used throughout the code to be declared here, this will decrease the total memory and free up space as there are less declaration of repeated variables. Throughout the program repeated code can occur this can cause memory to be clogged up which in turn wastes system resources this could have been used in higher priority processes. This can be easily solved by creating sub functions to be called as this reduces space in memory as well as in the code.

Improvements/extensions

Scoreboard

By using firebase without any errors disrupting the system for saving the scores onto a real time database, I would be able to illustrate and save the values onto the scoreboard screen. This feature would give a visual representation to the user about what other scores there are therefore creating a more competitive feel to the game. In the scoreboard screen the scores saved on the database will be retrieved and saved onto an array list of string, this will then be written and presented onto a list view in the ".xml" view. [1]

Level Creation

The user level creation is remarkably simple as the game prompts the user to input an integer to spawn several objects at random positions into the game, this can be further developed by allowing the user to place objects at specific positions throughout the map. This allows the player to create intricate map designs, and which will in turn make the game more challenging. The player should be allowed to place obstacles as well as bombs in specific locations, the user should be able to select the objects with a tool selector this will increase the ease of access to perform level creation.

Achievements

To add an aspect of gratification which the user will be rewarded after completing a task or collecting enough points, the game shall reward the user with achievements and allow them to buy skins for the snake and game. This will create a sense of accomplishment as the user will strive for a better score while trying to earn better skins, the skins will be accessed upon the user collecting enough points tallied throughout the lifetime of the game, this data will be used to purchase skins.

Conclusions

The “Snake” game uses several OOP techniques to help aid in implementing several features in the game, as this allowed classes to be separated and accessed on without interfering with one another, this in turn provided efficient coding without many errors or flaws occurred during development. Many features could be added on to make the game “complete” however these were not fully implemented, features such as the scoreboard using the real time database would be essential for creating a competitive game. Abstraction and inheritance were not fully added into the code as the theory was known but not correctly used when tackling the code, this would have created a simpler and clearer code to read and understand if the correct OOP techniques were used in the correct format.

References

1. [Firestore Realtime Database | Firestore Documentation \(google.com\)](#)