Module Code: CS2JA16
Assignment report Title: Android Game
Student Number: 26011251
Date (when the work completed): 17/04/20
Actual hrs spent for the assignment: 30+

Colour Dodge

A Game by Sam Dyas

Abstract:

Android libraries and functions have been successfully used to create a simple yet effective mobile game. Attempts have been made to utilise multithreading and memory optimisation. The project has strengthened knowledge of Java and allowed for a basic understanding of the Android Studio environment.

Introduction:

The game involves the player changing the horizontal direction a ball moves in order to navigate a series of descending obstacles, this is done by tapping and holding down on the screen. Each obstacle passed adds a point to the player's score. Upon reaching a score of 20 the difficulty will increase as the obstacles begin to continuously move horizontally. The game does not have a set end point and will continue until the ball collides with an obstacle. The final score will then be shown, and the player will have the opportunity to try again.

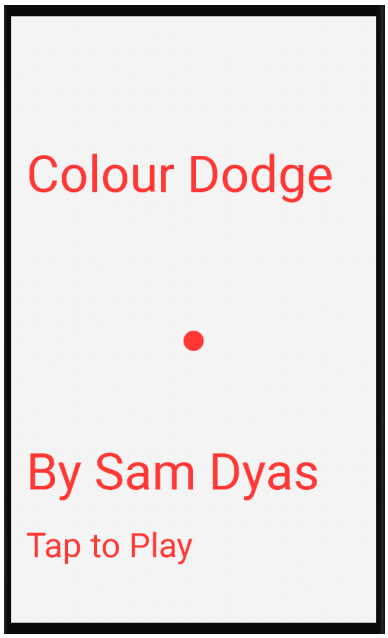


Figure.1 Start Screen

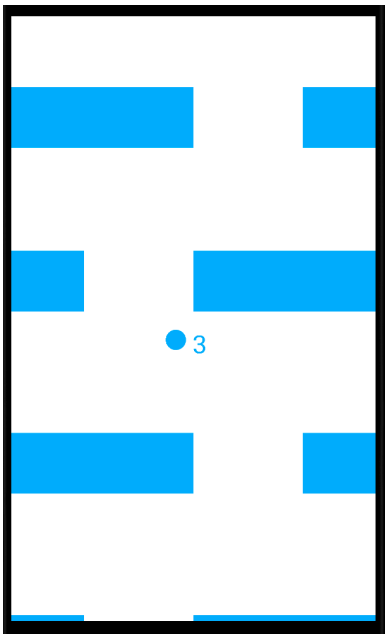


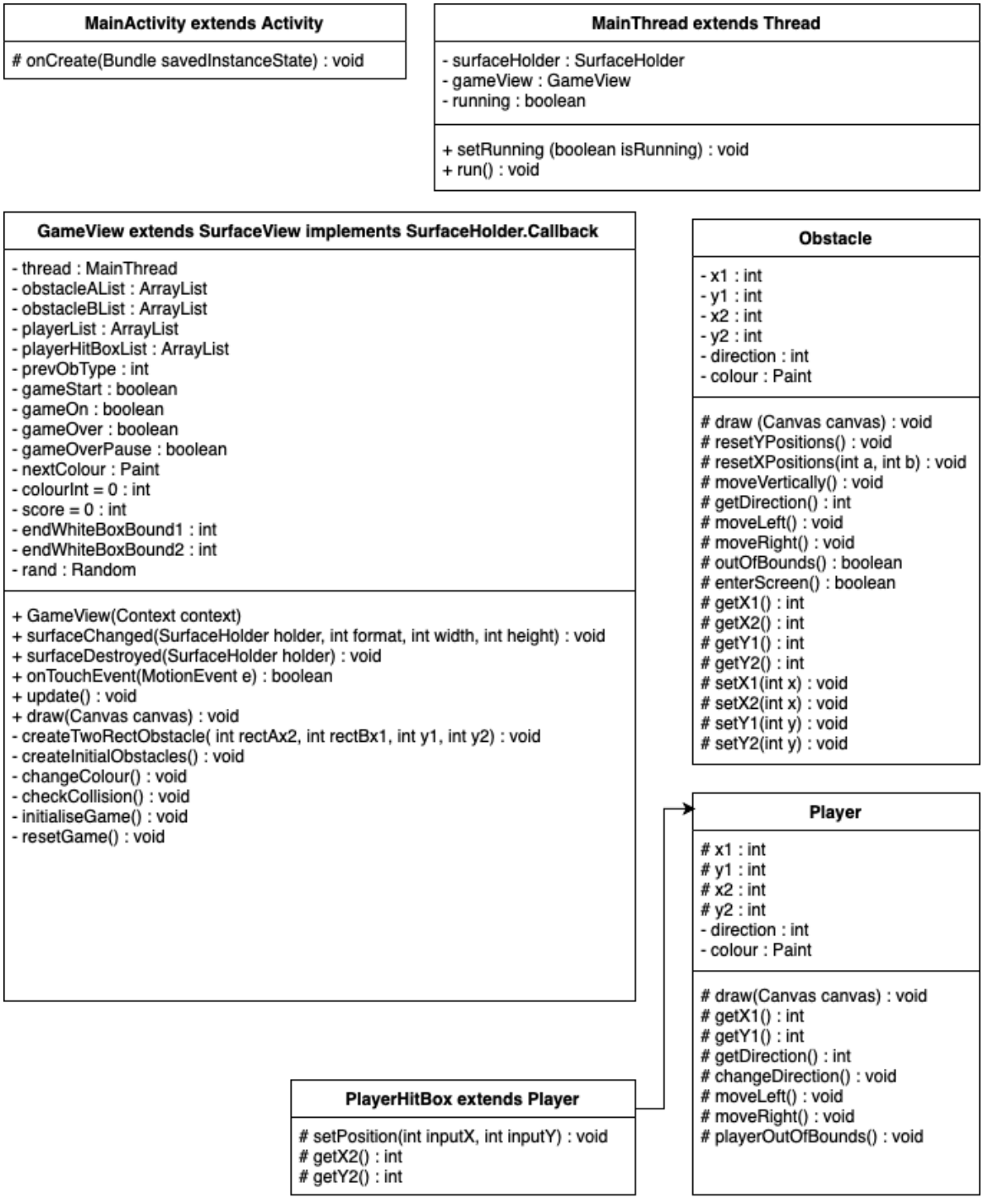
Figure 2. Gameplay

Figure 3. Game Over Screen

Start Screen:
Each screen is accessed via changing boolean values: gameStart, gameOn, and gameOver. When the game is initialised the gameStart boolean will be true, thus the start screen will be shown. The canvas will draw different objects depending on which boolean is currently true. Furthermore the onTouchEvent has four different outcomes, one for each game screen and a special fourth.

Gameplay:
When the gameOn boolean is true the update function will update the status of each object. Four obstacles (eight rectangles) are initially created and will move down the screen. Upon reaching the bottom their positions are reset to put them back at the top. The score text follows the player ball and will update accordingly. The ball has an invisible hit box rectangle to allow for collision detection between it and the obstacles. If the score is >=20 the update method will move the obstacles horizontally, alternating between left and right at regular intervals. As the ball passes each obstacle, the colour of the objects will change, cycling between the seven colours of the rainbow.

Game Over
When reaching a game over the gameOverPause boolean is used to prevent the game from detecting an onTouchEvent if the player still has their finger held down upon the screen.



Object-Oriented-Programming Design

The game code comprises of six classes: GameView, MainActivity, MainThread, Obstacle, Player, and PlayerHitBox. The GameView class creates and uses a MainThread object. The MainActivity sets the contentView to GameView. Eight obstacles are created: four for the left and four for the right; they are stored within their respective ArrayLists. One Player object and one PlayerHitBox object are created. Though Player and Obstacle share the same variables, they use enough unique methods to justify creating two separate classes as opposed to having one class with many methods. PlayerHitBox inherits from Player as it needs to utilise methods used by the Player object.

Memory usage and Speed improvements

Instead of using Bitmap images, the items within the game are created upon a canvas using the graphics libraries. As this can be memory intensive, efforts have been made to reduce the number of objects created and drawn upon the canvas. Only eight obstacles are created. Instead of deleting and recreating them upon going out of bounds, they are repositioned instead and reused. Only one player object is created and drawn, the hit box is not drawn, thus there are only nine simple objects present on the screen (not including text). Furthermore the graphics of the game are kept simple and as such do not exert a great demand for rendering. When restarting after a game over the objects are once more repositioned and redrawn.

Improvements/Extensions

I have attempted to use threading in order to increase the performance of the game. There is only one thread created however (MainThread) so it does not increase performance as much as if multiple threads were used. Further implementation of threading would be the focus of improvement in further iterations of the game.

Each tick-box represents one mark unless otherwise specified.		Range
Code style (2 marks each points): Following Java code conventions for all the project [yes] variable and class names [yes] method names [yes] indentation rules [yes] Using inline comments frequency [yes] Javadoc comments for every function (except getters/setters)	Overall OOP design and API usage: [yes] Appropriate use of inheritance (2 marks) [no] Appropriate use of Abstract classes (2 marks) [yes] Use of Android API classes/ methods (Other than in base code) (3 marks) [yes] Greater than 3 original classes (3 marks)	0-20
Functionality: [yes] On-screen menus (1 mark) [yes] Scores (1 mark) [yes] Controllable character (1 mark) [yes] Sensor interaction (touch/ accelerometer/ etc) (1 mark) Game has levels [ ] Works (4 marks) [yes - difficulty increases after a score of 20] Attempted (2 marks) Use of standardised data structures (e.g., List, Vector, Collection, Date): [yes] Works (10 marks) [ ] Attempted (5 marks) Randomly/procedurally generated features: [yes] Works (10 marks) [ ] Attempted (5 marks) Opponents (AI): [yes] Works (5 marks) [ ] Attempted (3 marks)	Design quality (both game and other game screens) (1 mark each): [yes] Professional looking [yes] Understandable game flow [no] Feedback to user instead of crashing, or recover [yes] Runs smoothly without interruptions [yes] Installs without error [yes] Starts/exits without error [yes - doesn't crash] Program does not crash [Not sure what this means] Produced in video form	0-40
Improvements marks: Online high score list [ ] Works (10 marks) [no] Attempted (5 marks) Multithreading improvements [ ] Works (10 marks) [Yes] Attempted (5 marks) Memory optimisation [ ] Works (5 marks) [Yes] Attempted (3 marks) User Level Creation [ ] Works (5 marks) [no] Attempted (3 marks)	Other extension/improvement / innovation [ ] Works (10 marks) [no] 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

PlayerHitBox class inherits from Player class.

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

Abstract classes not used.

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

onTouchEvent method found in GameView class.

Use of SurfaceHolder and SurfaceView within GameView and MainThread.

Use of android graphics libraries and custom colours (found in colours.xml)

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

Objects stored in ArrayLists

```
private ArrayList<Obstacle> obstacleAList = new ArrayList<>(); // arrayList for A obstacles
private ArrayList<Obstacle> obstacleBList = new ArrayList<>(); // arrayList for B obstacles
private ArrayList<Player> playerList = new ArrayList<>(); // arrayList to store player
private ArrayList<PlayerHitBox> playerHitBoxList = new ArrayList<>(); // arrayList to store the player hit box
```

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

```
private Random rand = new Random(); // random number generator
```

Random generation found within GameView, used to determine obstacle type i.e. middle gap, left gap, or right gap.

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

The obstacles call methods within update() in GameView to autonomously move and reset their parameters, including position, obstacle type, colour, and horizontal direction.

- Example of Online Scoring

Online scoring not implemented.

- Multithreading improvements

Attempted use of threading found within GameView class and MainThread class.