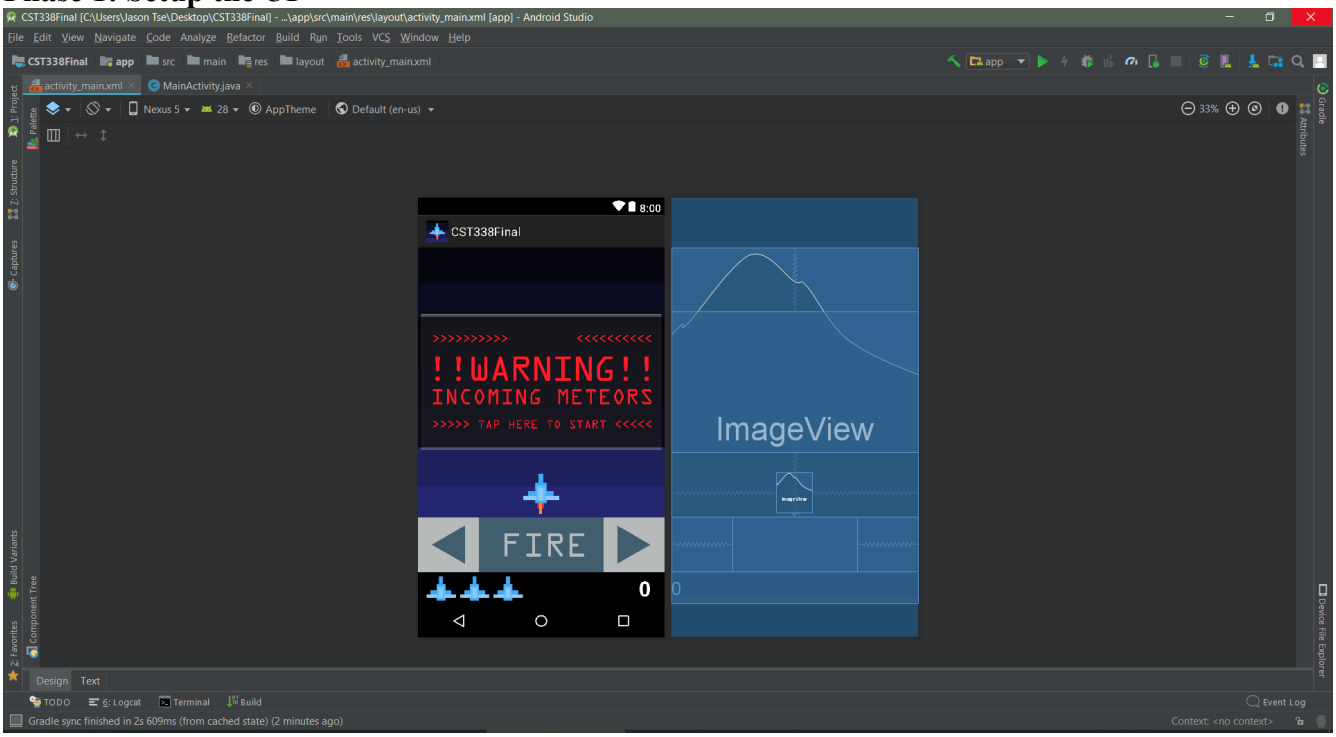
Jason Tse
CST 338
Final Project
7 August 2018

Final Project: Simple Android Game – Spaceship Arcade
This final project is a simplified arcade game where the player will dodge meteors and shoot lasers. There are 3 sizes of meteors that will fall at a random speed. Larger meteors require more hits to destroy than smaller ones. As your score increases, meteors will spawn quicker and fall faster.
The implementation is an Android application with a GUI and Multi-Threading.

Phase 1: Setup the UI



The game should have three major sections: the play area where the game objects interact, the control area for user manipulations and the game status area that displays the score and remaining lives. Optionally, the game can run in full screen, so overriding the onWindowFocusChanged() method to call a control hiding method can help to maximize the available screen area. We would also need to define a function for temporarily showing the default controls, as to not trap the player in the game.

We'll layout the following areas on screen:

- **The play area** should have a background image, a start button to trigger the start of the game, and a ship at the bottom of the play area. We could spawn the ship through code, but since the ship is only going to be moving side to side, using XML properties to dock the ship at the bottom can save us some lines of coding.
- **The controls area** should have three buttons: two for moving the player's ship left and right and a button for firing lasers.
- **The player status area** should have a graphic showing remaining lives and text for the score accumulated from blasting meteors.

We'll include the following variables, as we will need them for manipulation later on:

- **ImageView thePlayer –** Binds to the player's ship.
- **RelativeLayout theGameArea –** Binds to the layout containing the ship.
- **TextView playerStatusArea** – Binds to the status area.
- **public final static int FRAMES_PER_SECOND** – Determines how many frames will render per second. A higher number will have smoother graphics but more processing. A lower number will have less processing but have more stepped animations. 20 worked well enough for me, but a lower number is easier for debugging.

Phase 2: Setup the player controls

In the main XML, we can specify what the user control buttons do by specifying the android:onClick property of each button element to call a function in MainActivity.java. These functions must return void and take a view parameter of the calling button.

We'll include the following variables in the Main Activity java file:

- **final int PLAYER_MOVEMENT_SPEED –** Defines how many pixels the ship will move with each tap of the movement buttons.
- **final int MAX_LASERS –** Defines how many lasers can be on screen at the same time.
- **Laser[] lasers = new Laser[MAX_LASERS] –** Holds all spawned laser objects.
- **int gameWidth, playerWidth –** We'll need these variables for limiting the player from moving off the screen.
- **float playerX –** Same as above.

We'll define the following methods:

- **public void movePlayer(View view) –** This method executes player's the movement by determining which movement button was pressed using its resource id. Movement should only happen when it's within the bounds of the screen width.
- **public void fireLaser(View view)** – This method creates a new Laser object in a null index or the index of an inactive laser in the lasers[] array. It should also call to start the laser animation thread.

We'll create the following class:

- **Laser extends Thread** – An instance of a laser. A constructor should output the laser to screen, while the run() method will animate the laser.

With the following variables:

  - **private static final int LASER_SPEED** – Determines how far a laser can travel per frame.
  - **public ImageView image** – Holds the visual of the laser.
  - **public boolean isLive** – Indicates the status of the laser.
  - **private boolean onScreen** – Used to determine if the laser is still being shown on screen
  - **private Handler handler** – For making changes to UI in a safe manner.
  - **private int fps** – Holds the game's FRAME_PER_SECOND variable for animation.

And the following methods:

  - **public Laser(ImageView thePlayer, RelativeLayout theGameArea, int fps)** – The constructor. Should store the necessary parameters to local variables and display the laser on screen centered and above the player's ship.
  - **public void run()** – This method animates the laser while the laser is still live or on screen. This method should also pause the loop accordingly as per the frames per second specified. If the laser goes off screen or is no longer live, the laser can be destroyed by setting the

necessary flags, obtaining the contextual container using image.getParent(), and the removeView() method.

!!-IMPORTANT-!! All manipulation done to the UI via a new thread should be done in a new, anonymous Runnable object that is contained in a handler.post() method. This is because only the original thread (the UI thread) can manipulate UI. Any UI modifications done in a separate thread may throw errors or make the app unresponsive by blocking the UI thread. The handler object assists in queuing the manipulation to the UI thread in a non-blocking manner.

Phase 3: Create obstacles and game logic
On the click of the start button, the start button should disappear and start spawning obstacles. At the same time, the game should actively start collision detection checking the obstacle against the lasers and the ship. When the game ends meteors should disappear and "Game Over" appears.

We'll need to add one more method to the Main Activity java file:

- **public void startGame(View view)** – This method will remove the start button from the play area, instantiate a new Game object, and start the game by calling Game.start();

And we will also create two new classes, the Game class and the Meteor class. The Game class will handle the spawning of Meteor obstacles and polling for collisions. The Meteor class will be the obstacle itself and, like the Laser class, will handle output to the screen.

The Game class will have the following:

- **private ImageView thePlayer** – Holds reference to the player's ship.
- **private RelativeLayout theGameArea** – Holds reference to the layout containing the ship.
- **private TextView playerStatusArea** – Holds reference to the player's lives and points.
- **private int fps** – Holds a copy of the game's FRAMES_PER_SECOND.
- **private boolean gameOver = false** – Used for breaking the game loop when player's lives reached 0.
- **public static final int MAX_METEORS** – Setting for maximum number of meteors allowed on screen at any given time.
- **public static Meteor[] meteors** – This array holds all Meteor objects spawned.
- **private int frameCount, targetFrameCount** – These variables determine when the next meteor is spawned.
- **private Random random** – Randomizer to be used with spawning meteors.
- **private int playerScore, playerLives** – The player's variables. Score will be used to adjust difficulty of the game.
- **private Handler handler** – For handling UI manipulations
- **private Laser[] lasers** – Holds reference to the laser array from Main Activity java file.
- **public Game(ImageView thePlayer, RelativeLayout theGameArea, TextView playerStatusArea, Laser[] lasers, int fps)** – The constructor. This method saves parameters to local instances and initiate the game variables. For this game, player's lives will start at 3.
- **public void run()** – This method is the heart of the game. Every frame, this method will spawn meteor if a certain number of frames has passed, check if meteors collide with the ship, check if meteors collide with lasers, update the UI, and check for game over. The number of frames required to spawn a meteor will be random and augmented by the player's score.
- **public boolean isCollide(ImageView object1, ImageView object2)** – Given two images, this method will derive the images' bounding box and return true if those boxes overlap.

- **private void reduceLives()** – This method modifies player's lives count and updates the UI to reflect the current number of lives.
- **private void scoreUpdate(int scoreChange)** – Similar to the last method, but with the player's score. Currently 1 is always passed as the amount for the score to change, but this can be adjusted to give the game more difficulty.

The Meteor class will contain:

- **private int meteorSpeed** – Determines how many pixels the meteor moves with each frame
- **public int meteorHP** – Determines how many hits it will take to destroy a meteor.
- **public ImageView image** – Holds the visual for the meteor.
- **public boolean isLive** – Flag for determining if meteor is still in play.
- **private Random random** – Used for meteor's speed and size.
- **private Handler handler** – For making changes to UI.
- **public Meteor(final ImageView thePlayer)** – This method spawns a random size meteor and sets its health points and speed. To place image into the game area, use the player's ship that has been passed as a parameter to give the handler access. This is done using the getContext() method and passing that to the handler's controller. This will be done for all UI manipulations in the Meteor class.
- **public void animate(final int score)** – This methods moves the meteor. This method also removes the meteor from the game area if the meteor's health points drops to zero or the meteor goes off the screen.

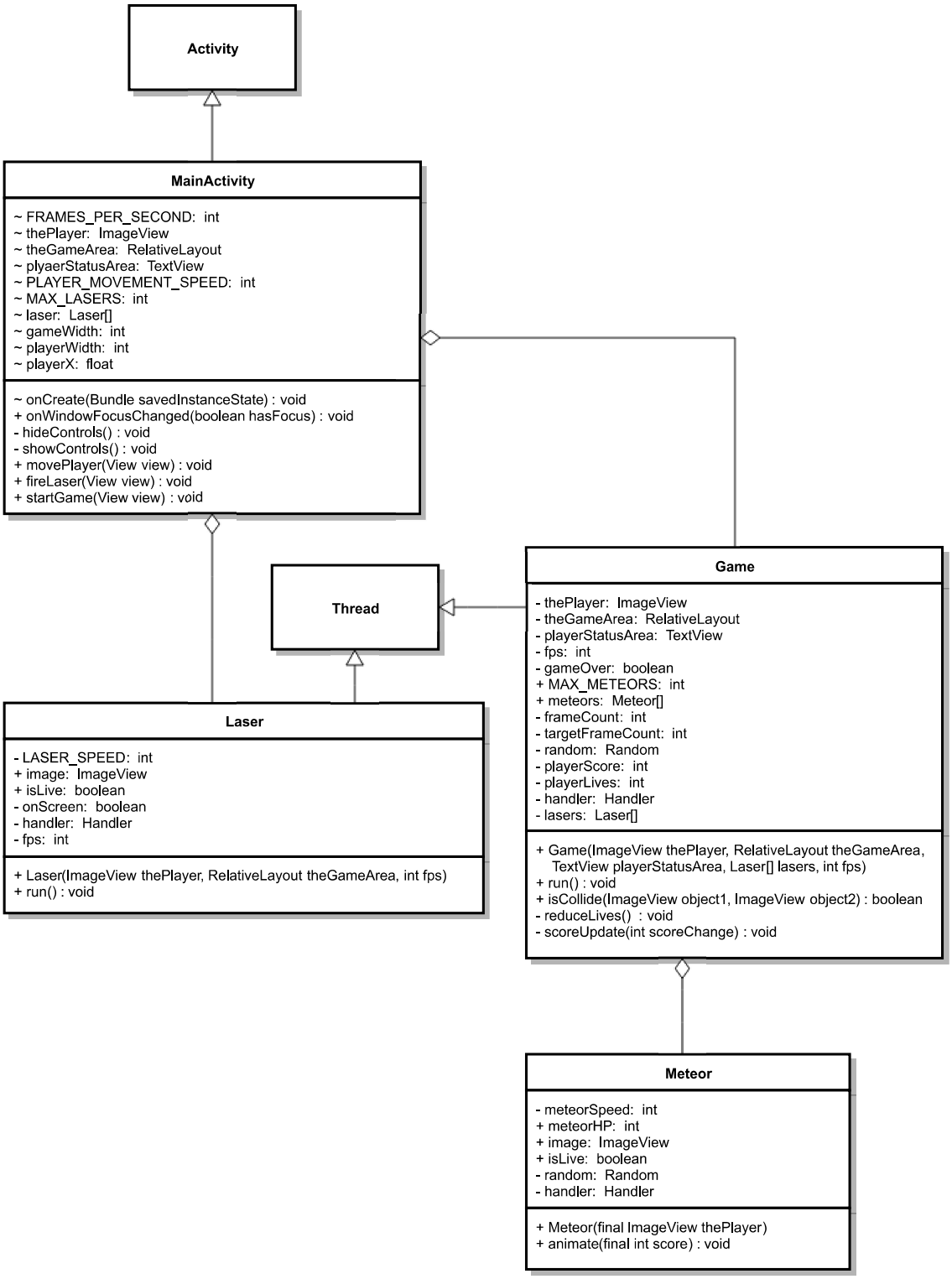
Phase 4: AVD setup and Testing

To run this application, the minimum API is 21 (Lollipop). All my testing was done in a Nexus 5, but this application should scale to most mobile devices.

For testing I did the following:

1. Rapidly press control buttons as a stress test. (Looking for UI thread blocking)
2. Move to the far left and far right to test ship movement range.
3. Fire as many lasers as possible to test MAX_LASERS
4. Shoot one of each size meteor to ensure correct health point is assigned to each meteor (1 for small, 2 for medium, 3 for large, 1 point for each successful hit)
5. Get hit by 3 meteors, watching the lives remaining graphic decrement. When game ends meteors should disappear and "Game Over" appears. Player's ship can still act but lasers will have no effect.
6. Swipe up from bottom to reveal controls, switch focus away from game, and return. Game should not have reset, paused, or crash.
7. Play game as high as possible to test play-through. If cannot reach a high score to test, change player starting score in the Game class to test stability. I've tested 100 and 200 point score, and game play was stable. 1000 point score was unstable and crashed, but probably not achievable anyways. 200 points is also not likely to be achievable.

UML Diagram


Activity
MainActivity
~ FRAMES_PER_SECOND: int
~ thePlayer: ImageView
~ theGameArea: RelativeLayout
~ plyaerStatusArea: TextView
~ PLAYER_MOVEMENT_SPEED: int
~ MAX_LASERS: int
~ laser: Laser[]
~ gameWidth: int
~ playerWidth: int
~ playerX: float
~ onCreate(Bundle savedInstanceState) : void
+ onWindowFocusChanged(boolean hasFocus) : void
- hideControls() : void
- showControls() : void
+ movePlayer(View view) : void
+ fireLaser(View view) : void
+ startGame(View view) : void
Thread
Game
- thePlayer: ImageView
- theGameArea: RelativeLayout
- playerStatusArea: TextView
- fps: int
- gameOver: boolean
+ MAX_METEORS: int
+ meteors: Meteor[]
- frameCount: int
- targetFrameCount: int
- random: Random
- playerScore: int
- playerLives: int
- handler: Handler
- lasers: Laser[]
+ Game(ImageView thePlayer, RelativeLayout theGameArea, TextView playerStatusArea, Laser[] lasers, int fps)
+ run() : void
+ isCollide(ImageView object1, ImageView object2) : boolean
- reduceLives() : void
- scoreUpdate(int scoreChange) : void
Laser
- LASER_SPEED: int
+ image: ImageView
+ isLive: boolean
- onScreen: boolean
- handler: Handler
- fps: int
+ Laser(ImageView thePlayer, RelativeLayout theGameArea, int fps)
+ run() : void
Meteor
- meteorSpeed: int
+ meteorHP: int
+ image: ImageView
+ isLive: boolean
- random: Random
- handler: Handler
+ Meteor(final ImageView thePlayer)
+ animate(final int score) : void

Screenshots