Embedded Game Specification
Vector Assault

Eric Klukovich
CPE 680 – Embedded Game Design
University of Nevada, Reno

November 3rd, 2015

1.0 Game Description

The game that is being developed has finally been named as Vector Assault. This section briefly describes the inspiration for Vector Assault and the details about the gameplay and other functionality.

1.1 Inspiration

Vector Assault is inspired from Geometry Wars 3, a top down 2D/3D game where the user controls a ship and destroys enemy geometry by shooting them. The main goal is to destroy as many enemies and accumulate as many points as possible. When an enemy is destroyed, a green diamond drops and if the user moves over it, then the score multiplier is increased. The player has a limited number of lives and the game ends when either the time runs out or the player has no more lives. An example screenshot of Geometry Wars gameplay is shown in figure 1.

Fig. 1: An example of Geometry Wars 3 gameplay

1.2 Gameplay and Functionality

This section discusses Vector Assaults gameplay and other functionality in more detail as well as covers the game's objective and rules, game modes and levels, controls, scoring system, and enemy types.

1.2.1 Objectives, Scoring, and Rules

When the game first starts, the player's ship is spawned in the middle of the level and the main objective is to score as many points as possible in short time frame. The player must destroy different types of enemies by shooting at them and if an enemy is destroyed, then a small green diamond drops. If the player moves over the diamond, within a certain amount of time, then the score multiplier is increased. Each enemy type has a set score value, making it the player's goal to kill enemies and collect as many green diamonds as possible to get the highest score. The player can move around throughout of the level but cannot move past the boundary. If the player hits the wall, then they can travel along it without any consequences. If an enemy hits the wall then then bounce in the other direction, or move along the wall towards the player.

The player is given three lives and will lose a life if the player and an enemy collides. As time goes on, the number of enemies that spawn will increase and the game gets progressively harder. The game will end if the player either runs out of lives, or if the time runs out.

1.2.2 Game Modes and Levels

Vector assault currently has one playable level and game mode. The implemented level is a simple two dimensional rectangular playing field where the player can freely move throughout it. At least one other additional level will be implemented that has obstacles in it so it creates more of a challenge to the player. Other two dimensional shapes, such as circles, can be utilized to create different levels.

The game also will also have two playable game modes. The first and implemented game mode is a timed survival mode where the player has to kill enemies to obtain the highest score in a given amount of time. The second game mode is a simple wave progression style game mode, where the player has to kill so many enemies every wave and tries to progress as far as possible.

1.2.3 Controls

In order for the player to control the ship, there are two virtual joysticks that are drawn in the bottom right and left corners of the screen. The joysticks are implemented using two circles, a larger one to act as the base and a second circle to show the direction of the "stick". The bottom left joystick controls the ship's movement, while the bottom right joystick controls the ship's shooting direction.

1.2.4 Game Interface

Vector Assault's interface was designed to be the least obstructive as possible so the user can focus on controlling the ship. Figure 2 shows an example of the in-game interface. The top left of the screen displays the number of lives, then the number of enemies remaining, and a button to pause and resume gameplay. The top center displays the remaining time the player has in the current game. The top right of the screen displays the score and underneath it shows the current score multiplier. Finally the yellow circles at the bottom corners of the screen are the virtual joystick that control the ship's movement and shooting direction.

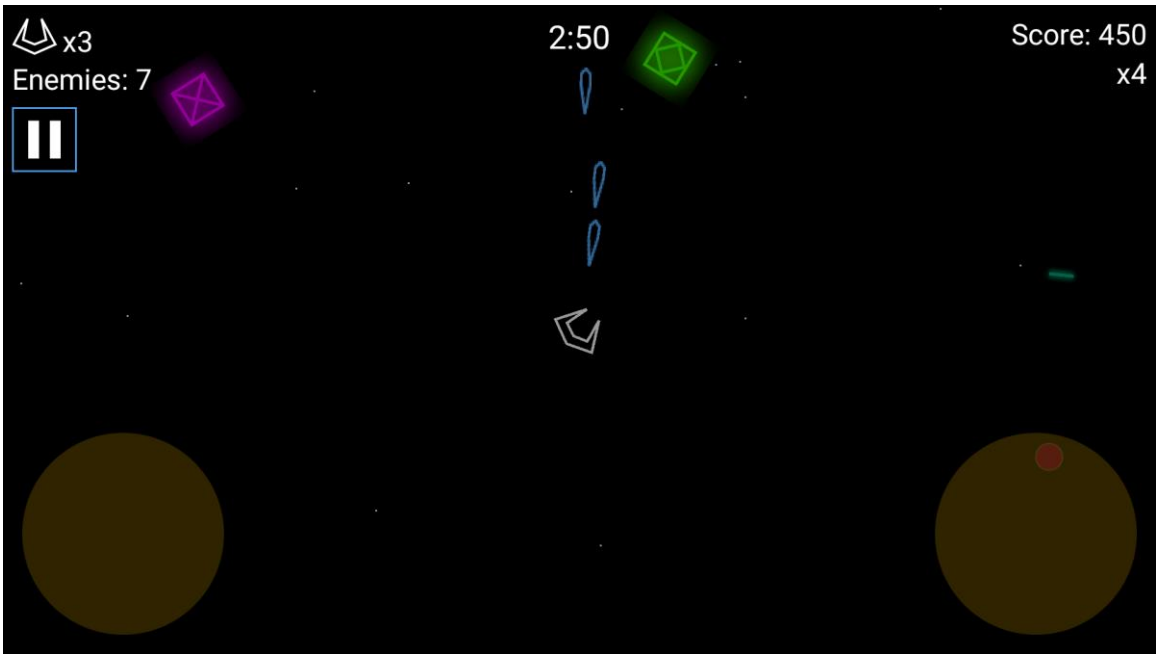

Fig. 2: Screenshot of Vector Assault's interface

1.2.5 Enemies Types and AI

Vector Assault has four main types of enemies and each enemy type has its own glowing sprite. The simplest type of enemy simply wanders around the playing field and acts as an obstacle that the player needs to eliminate. The next type of enemy will slowly move towards the player over time. Another type of enemy will move towards the player, but when it is killed, then smaller versions of the enemy will spawn. The final type of enemy will try to avoid the player's bullets by accelerating to the side of the incoming bullets. Other additional enemy types might be introduced depending if there is extra time. The simplest enemy is worth 25 points and the more advanced enemies will be worth up 100 points. The game will spawn more of the simpler enemies and then spawn the more difficult over time. The different enemies are shown in Figure 3.

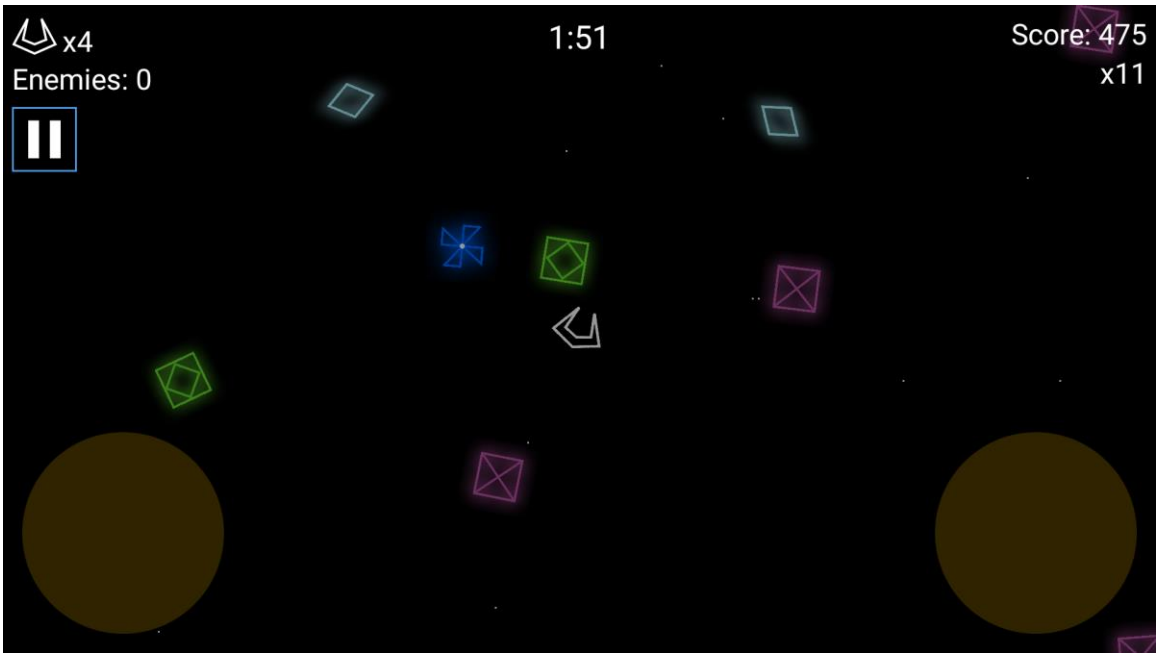

Fig. 3: An example of the different enemies

2.0 Design and Implementation

This section will discuss some of the design choices and implementation methods that are used in Vector Assault.

2.1 Platform and Toolchain

Vector Assault is developed for Android and uses OpenGL ES 2.0 to render the graphics. Android Studio 1.3.2 with Android 5.1 Lollipop Software Development Kit (SDK) is utilized for all of the game development. All of the main Android code is written in Java and the shader code is written in GLSL. A Samsung Galaxy S4 phone is used as the physical device for testing.

2.2 Implementation Goals

This section has two main parts, required goals and future goals. The required goals section describes all of the features that will be implemented by the final demo. The future goals describe possible features that may be implemented into the game if there is additional time before the final deadline.

2.2.1 Required Goals

- Full rendering engine that utilizes OpenGL ES 2.0
    - Render sprites

    - Particle system
    - Glow effects
    - Interface rendering
- Physics engine
    - Collision detection
    - Spring effect for game board
    - Change velocities/accelerations of objects
- Gameplay
    - Control system (virtual joysticks)
    - At least one fully functioning level
    - Two game mode
    - Scoring system
    - Power ups
    - Enemies
        - Different types
        - Different AIs

2.2.2 Future Goals

- Multiple Levels
- Multiplayer/Co-op
- 3D levels/enemies
- Multiple game modes
- Different difficulties

2.3 Implementation

2.3.1 Rendering Sprites

All of the sprites are created as a two dimensional PNG image and are imported into the game. Every sprite contains four points that create a quad and the image is then textured onto the surface of the quad. Alpha blending is enabled to create the transparency for each object. The glowing effects behind each sprite is added with Photoshop.

2.3.2 Particle System

The particle system for Vector Assault needs to be as efficient as possible in order to have a large amount of particles on the screen and have them update without lagging the game. In order to do this, the particle system is implemented as a circular array of particles in order to keep the memory allocated and it can always have a section of active particles. A new particle is added and if the index is at the end of the array, then it is moved back to the beginning. The indices are set up to always have a small range of particles that are active and the particles out of that range are not being rendered. Each particle has several properties such as life percent, color, speed, and direction to have them behave correctly. When a particle is created, all of the previously mentioned values are set randomly (restricted by max values) in order to create a pattern, such as an outward circle for the explosions. On every frame, the particles are updated based on the position and speed and

are redrawn to the correct locations and the life percent is decreased. When the life percent reaches zero, then the particle is marked as inactive and no longer rendered. Each particle is textured in order to make it look like a specific effect. Figure 4 shows an example of a bunch of particles acting as an explosion.

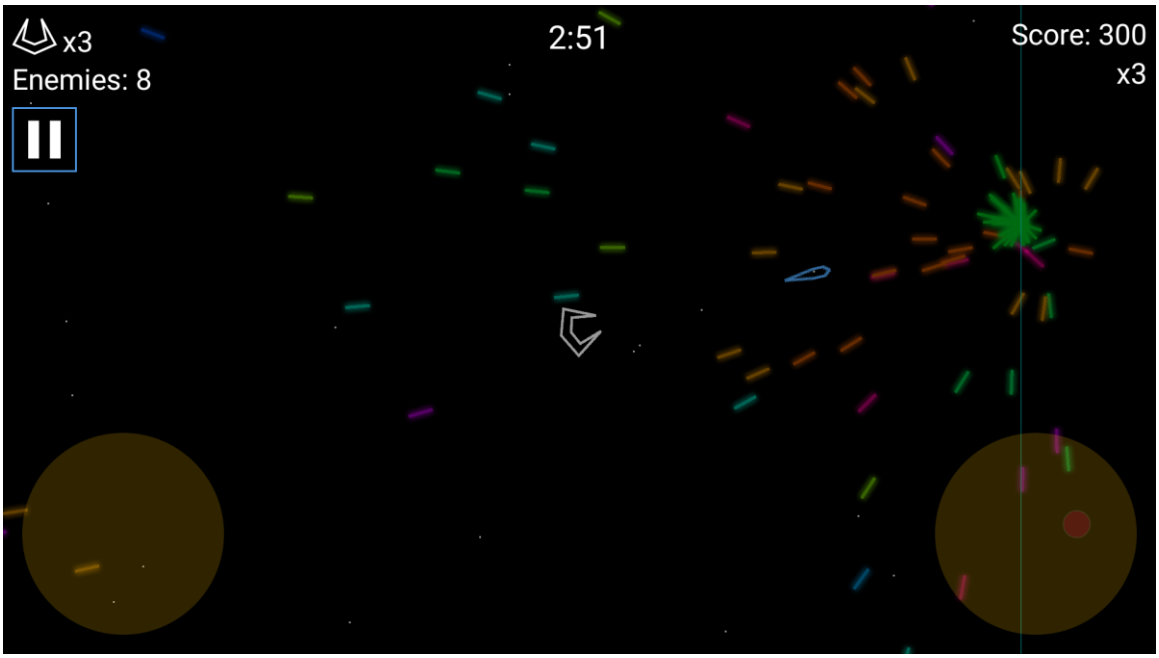

Fig. 4: An example of an explosion with a bunch of particles

2.3.3 Interface

Vector Assault uses OpenGL to render all of the graphics, and this requires a special view called a GLSurfaceView to render all the graphics to the screen. If only this view is used, then the native Android views (text, buttons, etc.) cannot be used. In order to make the native Android views work with OpenGL, a Frame Layout needs to be utilized to allow multiple views to be layered on top of each other. By layering the different views, buttons, text, and other layouts can be incorporated with the game. Vector Assault has three main layers and is shown in figure 5.

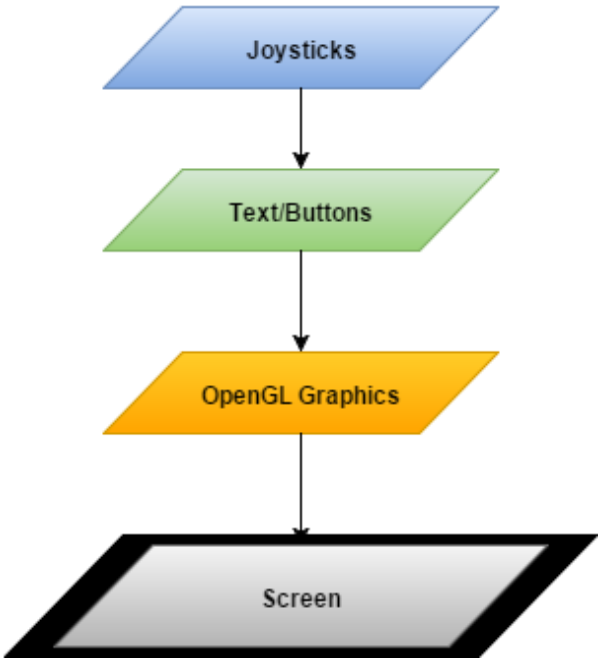

Fig. 5: Example of how the views are layered in Vector Assault

2.3.4 Controls

Android does not have built in virtual joysticks to be used for controls. As a result a custom view was created and a method to draw the joystick as well as control the player was implemented. The joystick implementation is based on an existing piece of unsupported code on Github, and uses two circles to behave as a joystick. The first circle is a larger circle that acts as the base for the joystick and limits the boundaries of the second circle. The second circle acts as the stick and can move around within the larger circle. A method was developed to determine the direction of the joystick and to move the player accordingly. Figure 6 shows an example of the joystick.

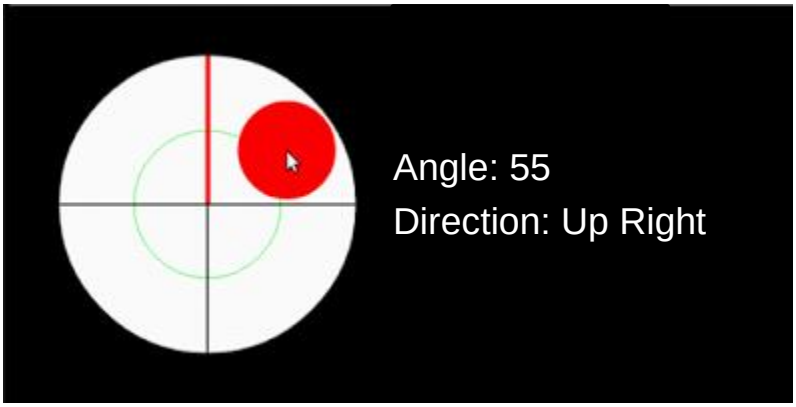

Fig. 6: Example of the virtual joystick

3.0 Game Changes

Overall, no major changes to the design or implementation have been made since the initial game proposal. One area that was changed was that Bloom effects have been removed as one of the goals because the way the sprites are rendered would give a square glow effect rather than a circular one. As a result, the glowing effect behind each sprite is created with Photoshop. Another change that was made was having to implement a custom text rendering system for OpenGL. Since an alternative method for using the built in Android views for text rendering was found, then the need to create the OpenGL text is no longer needed.