Universe Run
Final Report
June 2014

Computer Science 179N, Spring 2014
Professor Victor Zordan, Instructor
Calvin Phung, T.A.

Team Buzz Members
Jonathan An
Nicholas Collier
Danelze Strydom
Shayan Mehrazarin

Music By:
Aram Adajian

I. Introduction

A. Narrative (Shayan Mehrazarin)

In Universe Run, the player has to survive in space as long as possible. The objective of the game is to navigate the spaceship throughout space while dodging various dangerous obstacles. The player faces incoming asteroids along with a swarm of enemy ships. Besides dodging the asteroids, the player must also dodge the lasers that are being fired by the enemy ships.

Avoiding the asteroids and the swarm of enemy space ships can be chaotic enough, but the player is in luck. Embedded within the wave of asteroids and lasers being shot by the enemy space ships are various power-ups. The player's space ship can collect hearts, giving the player an extra life. The player can grab coins to boost up their score. Missiles can be picked up to enable the player's spaceship to shoot faster for a limited time. Clocks can be picked up to slow everything down in space except for the player's space ship.

B. Genre (Shayan Mehrazarin)

The setting of Universe Run includes elements that are associated with outer space and Sci-Fi. The gameplay of Universe Run can be classified under the Arcade and Action genres.

II. Core Mechanics & Unique Aspects

A. Core Mechanics (Shayan Mehrazarin)

The objective of the game is to navigate the spaceship throughout space avoiding various dangerous obstacles. These obstacles include asteroids, enemy space ships, and the lasers being shot by the enemy ships. However, there are various power-ups that are spawned among the obstacles, including hearts, coins, missiles, and clocks. The heart will increase the player's amount of lives by one, the coin increases the player's score count by one, and the missile allows the player's ship shoot faster. The clock, on the other hand, slows everything down except for the player for a short period of time.

The player's spaceship is able to shoot freely at anything, but not everything can be destroyed. The player cannot destroy the asteroids or any of the power-ups; only the enemy space ships can be destroyed if shot at by the player. The player does take damage if hit by an asteroid or laser shot by an enemy space ship, where one life will be deducted. The game will end if the user runs out of lives and the game over screen is presented to the user. The game gets more and more difficult as the player progresses, posing harder and harder challenges.

B. Unique Aspects of Universe Run (Jonathan An)

Universe Run is unique because it combines aspects of several different games. It is similar to Temple Run in the aspect that it is a time-based scoring game in which the player is continuously dodging obstacles to survive. It is similar to N64 Starfox because of the spaceship, shooting crosshairs, and barrel rolls. It is also similar to the arcade game Space Invaders because of the aspects of space and shooting enemies that move in a pattern. Universe Run is a simple to pick-up type of game that has easy controls and style of play. It can be easily seen as a smartphone application or arcade game.

III. Software Highlights

A. Technical Details (Jonathan An)

Universe Run is created in Unity3D and some models are made in Maya. The spaceship is modeled in Maya and also some asteroids that we did not end up using. All of the game's scripts are coded in C#. There are about twenty-five different scripts that we used to create the game. The game is developed for PC and can also be played on Unity's webplayer.

B. Key Scripts (Jonathan An)

The *shipBehavior.cs* script controls all aspects of the player's ship. It is moved using Unity's Input controller class to get the axis of direction from the WASD or arrow keys. It then uses this axis of direction and some added velocity to move the ship. This script also controls the rotation of the ship by using a LookAt() function which rotate the ship to look at the location of the mouse with a preset z-coordinate. It also controls the life variable.

The *enemyShipBehavior.cs* script controls the movement of the enemy spaceships. It gets a vector of waypoints by finding the game objects set in the scene. The game objects it finds and the waypoint pattern it follows is based on a random variable that is generated in the *enemySpaceshipSpawner.cs*. All of the game objects that form different patterns are empty game objects and are set up hierarchically in the game play scene.

The *destroyObjects.cs* script controls what happens when objects collide besides all the powerups. It uses OnTriggerEnter() function and detects which objects have collided. If an asteroid collides with the ship then the player loses a life and the asteroid is destroyed. A similar behavior happens when the player gets shot by an enemy or vice versa, except the enemy is destroyed.

The *Spawner.cs* script controls the spawning of asteroids and powerups. This script is all time-based. As time goes on, more asteroids spawn and less power ups. It includes our algorithm for deciding how often powerups should spawn. This script uses an IEnumerator() function to start a coroutine of instantiating objects.

The *Shooting.cs* script controls the player's shooting. It basically instantiates a bullet prefab in front of the ship and gets the ship's direction and adds velocity to the bullet in that direction. The *enemyShooting.cs* is similar except that it gets it's direction by using the LookAt() function to look at the player's ship.

The *PowerUpWeapon.cs, PowerUpLife.cs, PowerUpScore.cs,* and *PowerUpSlowTime.cs* scripts are all similar except each access a different variable from a different script. They make a slight change to the different variable to give the power up skill. It also detects collision with the ship and turns the doBarrel variable to true on the *shipBehavior.cs* script.

The *HUD.cs* raycasts from the player's x-coordinate and y-coordinate plus Vector3.forward in the z-direction. It does several raycasts from different parts of the ship to get a better detection for collision. It returns the smallest distance from each of the RaycastHit data type. Based on this minimum distance, the *changeHUDcolor.cs* changes the color of the HUD on the bottom left of the screen.

The *Score.cs* script just increases the score of the game based on time. The score increases by one point for every second that passes. The score can also increase with the score power up by twenty-five points.

IV. Technical Challenges (Danelze Strydom, Jonathan An)

The biggest challenge we faced was implementing AI the way we wanted it to work. While building the game we simply used enemy ships that moved through space along with the asteroid. Next we had the enemy ships spawn at random locations, but remain in a fixed position, and then rotate to face the player and fire. This was boring, so we decided to use different formations that the ships would appear in and shoot from. This worked well and we would have them spawn in these different formations and respawn as soon as each wave was destroyed. This presented the problem that each level progresses from hard to easy: there are many ships, and then taper down to none. The next step was then to move these formations and continuously spawn enemy ships. The previous methods had all created instances of enemy ships that could not be referenced to determine "where" each enemy ship was. So the final challenge was implementing a waypoint system. This created a path of points that the enemies would follow. We finally got it to do what we wanted, but it was a continuously changing process.

Another challenge we faced was working with sound effects. We wanted sound effects to play on powerups, but could not get this to work. It worked on other prefabs in our game, such as shooting and collisions. This was baffling and we followed many different tutorials and even asked classmates, but could not get it to work.

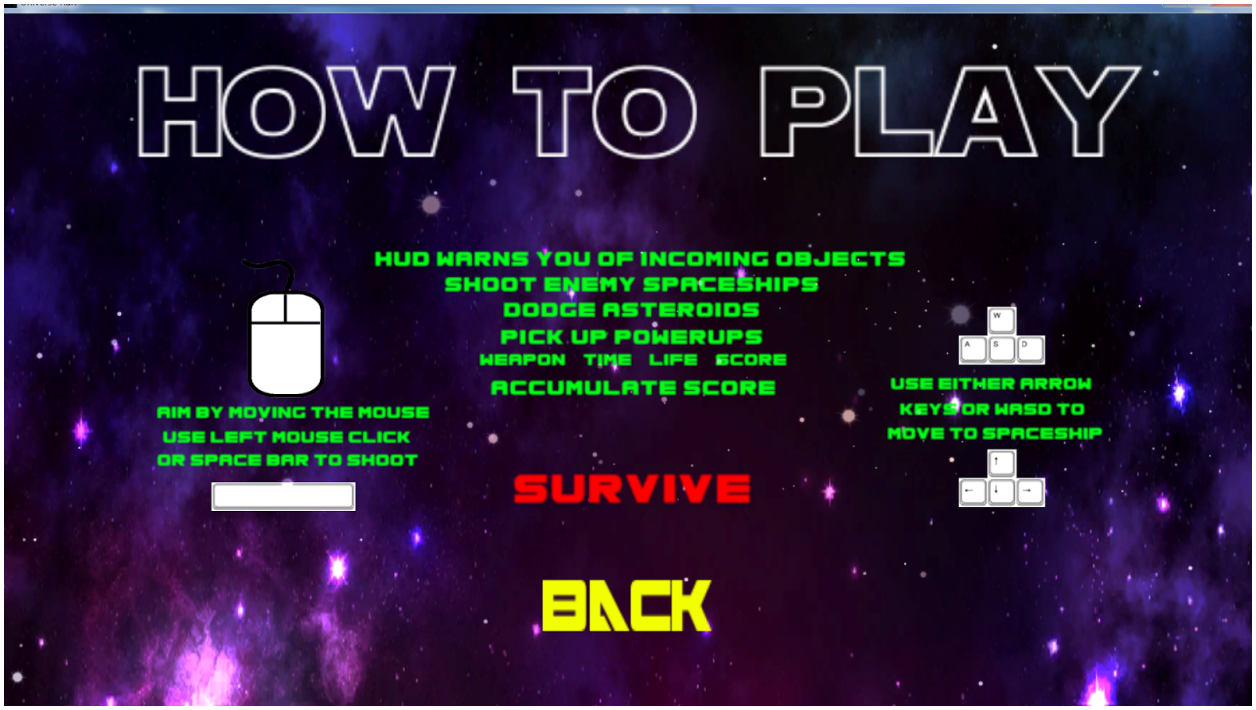
One of our first challenges that we worked on until the last week of the project was fixing the game's depth perception. It was hard to tell when the player was going to get hit by any incoming objects, especially the asteroids. One of our first fixes was to change our main camera's field of view. We then added a dark red fog color in the distance so that the objects would become a little brighter and change to it's normal color as it came closer to the player. Another addition to fixing the depth perception was moving the directional lighting to the bottom left of the screen pointing to the top right. This made it seem as if the sun was behind the player and to the left a little bit which we thought helped fix the problem more. Another fix we tried to

do was add a colored line that pointed forward coming from the front of the ship. This helped a lot in telling when an object was getting closer and if your shot was going to hit an object. However, it looked awkward and made the game less appealing. We decided to remove the line and add crosshairs. Two crosshairs were attached to the ship each at a different depth in the z-axis. This helped because you could tell when an object is passed one of the crosshairs and getting closer to the ship. Our final addition to this problem was adding a HUD that changed colors from green to yellow, and yellow to red as an object approached. The HUD detected the distances of an object through several raycasts from the player's ship and returned the closest distance. Based on that distance the HUD would change to a certain a color.

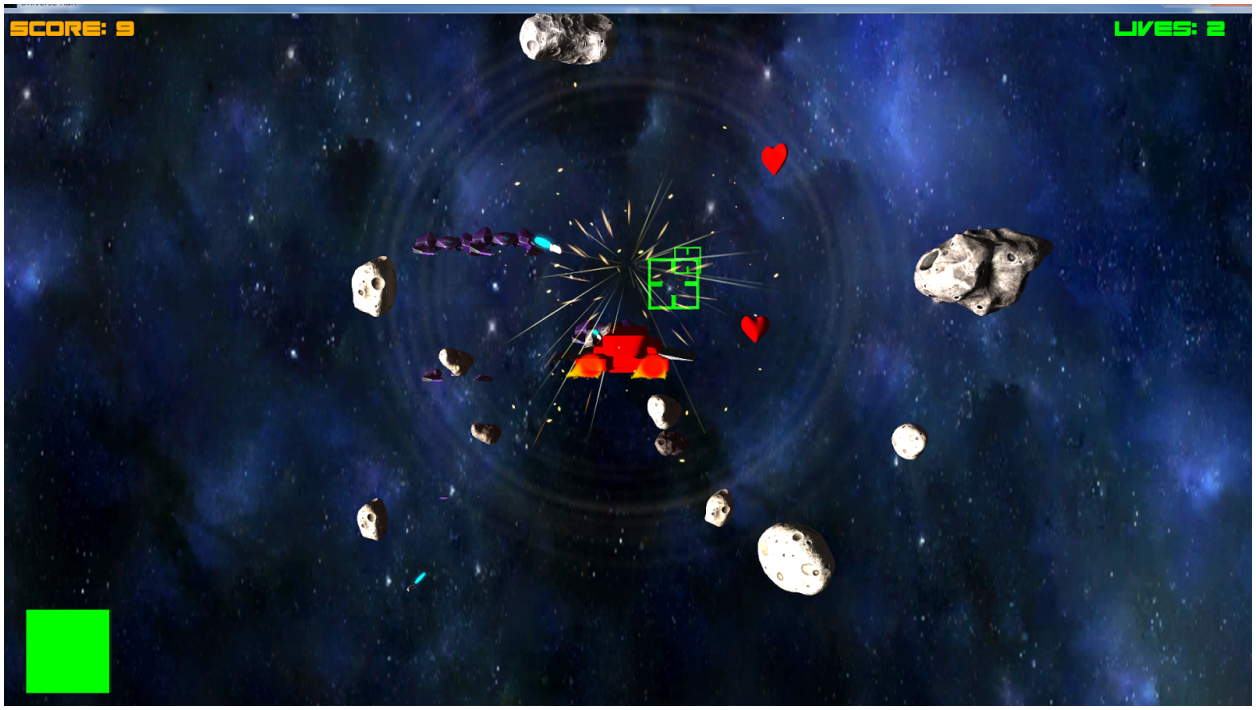
V. Final Game (Danelze Strydom, Jonathan An)

The final product of our game included all of our basic requirements and more. It contains all of the core mechanics of a simple video game and it is fully playable. The game includes a menu screen, how to play screen, gameplay screen, and game over screen. The game has several models including the player's spaceship, enemy spaceships, asteroids, heats, coins, missiles, and clocks. It has a fully implemented scoring and lives system. The AI system includes several formations and patterns that they randomly are generated and move in. The shooting system for the player includes a time-based offset shooting and rapid fire shooting. The enemy shooting is only a time-based shooting that continuously aims at the player's spaceship. The asteroids continuously spawn and move towards the player in a manner that makes the game harder as it goes on. There are four different power-ups and each help the player in a different way. The final product contains a soundtrack for the menu screen and a different soundtrack for the gameplay screen. There are shooting sound effects when the player gets hit, hits an enemy, or hits an asteroid. Below is a few screenshots of the game:

Start Menu



Game Instructions



Gameplay

Pause Menu

Game Over Menu

VI. Future(Nicholas Collier)

If we were to continue working on our project there would be a few pretty useful things that we would add in. One of the main things we would add in is a high score. This way players can see if they did better than they have before. We would also add in some sort of currency into the game that allows for different spaceships. This would add a little bit of incentive to continue to play the game. We would also do a lot to the background to make it feel and look better as you play. Right now, the background is very boring. It is basically a nonmoving picture. We would add in objects and animations to liven up the game. If we had a large amount of time we would add in wormholes or black holes that transfer you from one area of space to another. This would give a feeling of levels and progression. It would allow us to make varying space art styles that would spice up the entire game. With this different areas of space we could also change up patterns of spaceships and things to add some variety and more structure to the game. We could use these portals to increase the difficulty and show the players when the difficulty would be increased.

We would also aim to run it on android. This would not be too difficult with Unity's built in Android compilation. We might have to change a few things, but it would not be that hard. Android is a better platform for our game than PC. There are very high expectations for games on PC. If we could get it on Android, we could monetize the game and hope that it gets some downloads. We could also add in more sounds and animations to the game.

There is decent music and animations, but some things could use sounds, such as power ups. We could also use more animations. Animations just make the game feel more interesting and special. If we added other things we would also need to make sounds and animations for them too. One other thing we could add to our game is a multiplayer mode. This would allow players to have their friends in the game with them as well. It would make the game a lot more fun and add a new way to play.

VII. Close(Nicholas Collier)

Universe Run was a fun project to work on. It took a pretty big chunk of time, but it was nice seeing our game working and looking as good as it did at the end. We had a lot of problems that we had to overcome throughout the ten weeks. We tried as hard as we can to improve upon them. Some things we could not really fix, but others we did nicely. We definitely took all the input from others to shape our game. Our game was definitely not perfect, but it still had its positives. There are many things we can fix and work on, but the game in its current state is still a fun game to play. We are proud to produce a game in ten weeks as good as it has come out.

There were a lot of things that we learned over the ten weeks. We learned a lot about working as a team. Teams have some major problems that need to worked out. Our team had to learn to communicate and know when to work as an individual or as a team. We tried spending most of our lab time working to put all our stuff together. This seemed like a great method. It took a few weeks to learn that is how we were doing things. We also learned that sometimes working individually is not the best method. Some things need more minds than others. We also learned that Unity has a great community with a lot of available information. We learned how to deal with each other. We learned how to make a decent game in ten weeks.