ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

Space Shooter Game

[1] Ansari Aaqib, [2] Madhavi Jadhav, [3] Megha Sudke, [4] Vipul Gawad, [5]Vijaya Sagvekar, [6] Vinod Sapkal
[1] BE Student PVPPCOE, [2] BE Student PVPPCOE, [3] BE Student PVPPCOE, [4]BE Student PVPPCOE,
[5]Assistant professor PVPPCOE, [6]Assistant professor PVPPCOE

Abstract: **- In our project we will be placed in a space setting. The player can control a spaceship at the start of the game and can be changed during the game play. The player can move in 2 degree of freedom. There are three types of enemies like ships, asteroids and boss. The enemy ships will have basic Artificial Intelligent system as they can change their paths while shooting in the direction of the player. The asteroids will follow a straight line in any random direction. The boss will be a scaled up version of other types of enemies which will stay at the screen until defeated.**

Index Terms**—Artificial Intelligence (AI), OpenWorld, Cross Platform.**
Keywords**— Space shooter, Game, Top-down shooter, OpenWorld, Unity3d, Artificial intelligence.**

I. INTRODUCTION

The game must allow the player to play the game, save and load the progress at any time, have score system to rate player performance. The game will be divided into stages. The Player can roam in an OpenWorld when not on a mission. The player controls character movements over obstacles, defeat enemies, reaching end goal to finish one stage. Player character will loss a life or reduces its shields when collided with enemies or lethal obstacles.

II.GAME ENGINE

In our project we selected Unity3D version 5.6 for development of the project. Unity3D is a powerful cross-platform 3D engine and it isuser friendly development environment. Unity3D is a easy to understand so anybody who want to easily create 3D games and application for mobile, laptop, computer, web etc. create 3D games and applications for mobile, desktop, the web and consoles.

A. MOVEMENT OF GAME

A. Movement in Mission

Movements in mission is designed in such a way that confine the player to face forward towards the stage progression and control the space required to design the mission towards sure achievement if the player goes through all the enemies.

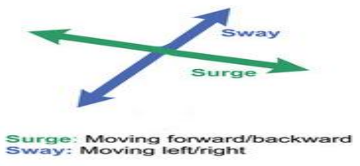


Fig 1: Movement in Missions

B. Movement in OpenWorld

Movement in OpenWorld is given that user can travel in any direction in a 2D plane.

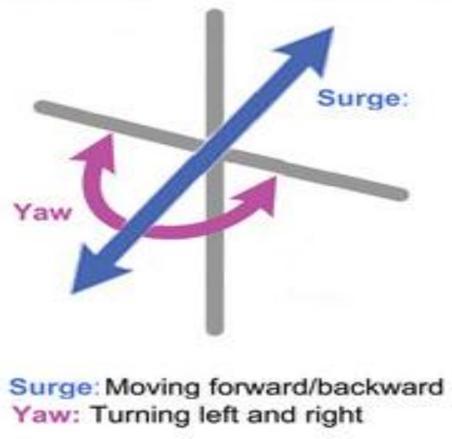


Fig 2: Movement in OpenWorld

III.ENEMY BEHAVIOUR

The enemies will be coded with basic artificial intelligence that can move towards enemies, some can ambiguity the attacks of the player and change their course midway on the playing area. The enemy can decrease player's health by either shooting at him or colliding with him. In either case the player's collider mesh will be affected by some other colliders.

A. Algorithm

The player (green) movement is restricted to the X and Y axis. Enemy (or enemies) is at the top of the screen, his movement is also restricted to the X axis with a fixed speed in Y axis to simulate movement towards the enemy. The player fires bullets (yellow) at the enemy.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 3, March 2018

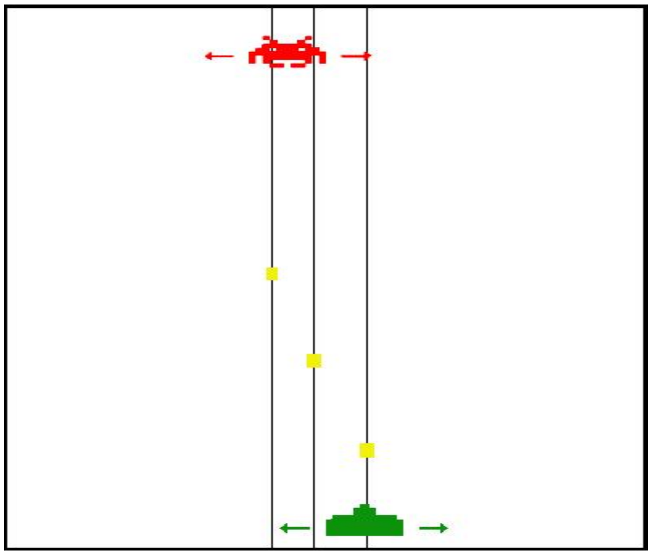


Fig 3: Enemy Movement Behaviour

The screen is divided into discrete sections(light grey and light blue) and assign weights to them. There are two weights: The "bullet-weight" (dark blue) is the danger imposed by a bullet. If a bullet exists in a section it will add 0.2 to the bullet weight. If multiple bullets is in the section the bullet weight will multiply. The second weight is the "distance-weight" (dark-green). The closer the bullet is to the enemy, the higher the "distance-weight" (0.1,0.3,0.5). Lanes without a bullet have a weight of 0.0.

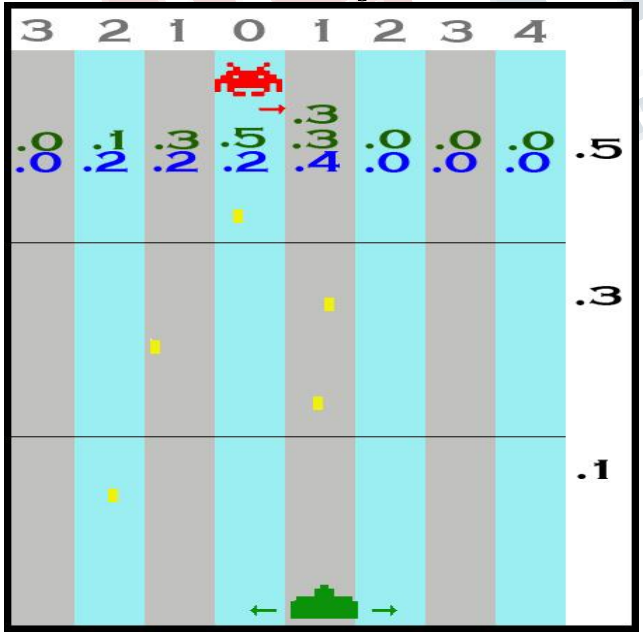


Fig 4: Enemy Movement Calculations

The enemy will decide where to move based on where it is located(Numbers in dark grey indicate its position). The section enemy is located in is given the distance as "0". the further away the section from the enemy is the distance is increased by "1".

The enemy will calculate the minimum "bullet-weight"+"distance-weight" and move towards the closest section available to this minimum value.
In the fig4. the enemy moves to the right.

A) OPENWORLD

In players will be placed in handcrafted OpenWorld environment which will make the player either travel the area or can go to the next mission area. The player can come across space stations or random trade ships to buy new missiles or ships from in-game money gained during and after completing a mission.

IV.COLLISISONS

In a game that focuses on shooting down enemies, or avoiding being shot down, a system to handle collision is vital. This section defines colliders, a way to detect colliding objects, our perceptive behind collision in a space-shooter game, results from the game, as well as a discussion about what could have been done differently.
Collider components state the shape of an object for the purposes of physical collisions. A collider, which is invisible, need not be the exact same shape as the object's mesh and in fact, a rough estimate is often more efficient and indistinguishable in game play. The simple (and least processor-intensive) colliders the so-called primitive collider types. In 3D, there are the three types collider like Box Collider, Sphere Collider and Capsule Collider. In 2D technique, we can use the Box Collider 2D and Circle Collider 2D

A. Using Colliders as 'Colliders'

The default setting for any collider attached to an object is to restrict the object being passed through be other world objects. The collision event must be handled by a script attached to one or both of the objects involved in the collision. Unity has a built in function for detecting collisions. This function in fig is attached to the Red Sphere.

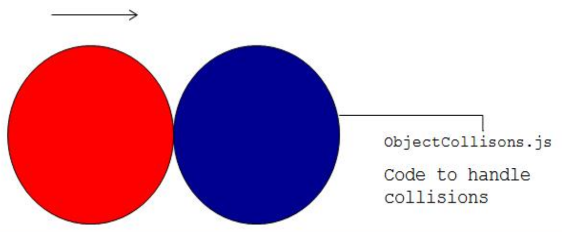
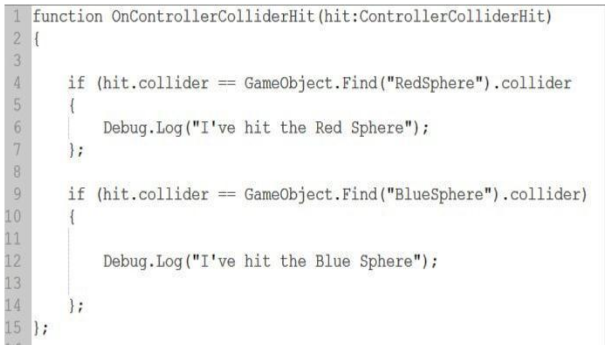


Fig 5: Colliders as Colliders Example



```
function OnControllerColliderHit(hit:ControllerColliderHit)
{

    if (hit.collider == GameObject.Find("RedSphere").collider
    {
        Debug.Log("I've hit the Red Sphere");
    };

    if (hit.collider == GameObject.Find("BlueSphere").collider)
    {

        Debug.Log("I've hit the Blue Sphere");

    };
};
```

Fig 6: Collider Implementation

B. Using Colliders as ‘Triggers’
Object collisions may be used to generate events ‘triggers’ which can be used to update logic in the World, generate actions, instantiate (create) new objects or remove unwanted objects from the world. Using triggers is one way for the player to collect items that update values via attached scripts.

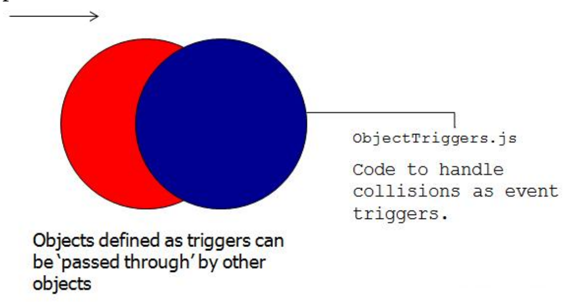
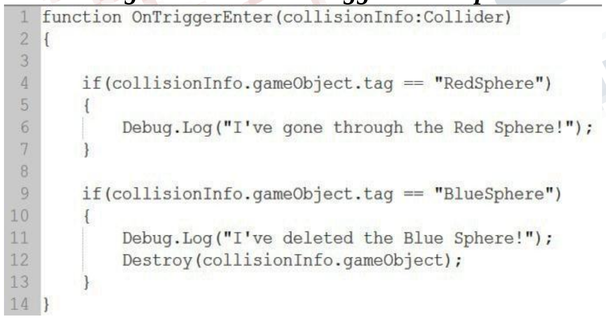

ObjectTriggers.js
Code to handle collisions as event triggers.
Objects defined as triggers can be ‘passed through’ by other objects

Fig 7: Colliders as Triggers Example



```
function OnTriggerEnter(collisionInfo:Collider)
{

    if(collisionInfo.gameObject.tag == "RedSphere")
    {
        Debug.Log("I've gone through the Red Sphere!");
    }

    if(collisionInfo.gameObject.tag == "BlueSphere")
    {
        Debug.Log("I've deleted the Blue Sphere!");
        Destroy(collisionInfo.gameObject);
    }
}
```

Fig 8: Trigger Implementation

V.LIGHT SETTINGS

There are many ways to represent light sources, common ones being point lights, directional lights, spotlights, and area lights. Both the spotlight, and the directional light sources have defined directions, so when the light source is in the middle of the plane, as in our game, they are not very suitable; an omni directional light source would be preferred. This leave the point light source and area light sources as likely candidates.

A point light source is a light source where all the light comes from an infinitely at a small point. This approximates lighting from a light source that comes from an infinitely either very small, or far away. However, when the light source covers a larger area, a point light source might be too imprecise, as it will not illuminate the surrounding objects as well as a large light actually would (see Figure 3). An area light source would solve this issue by simulating the larger area, commonly using multiple point light sources, which results in more proper lighting depend on the cost of computational power. One way to avoid this issue is to, for every fragment of the object on which to apply lighting, choose the best point of the light source, and use only this point, as a point light, for the calculating the light.

In the special case of a spherical light source shading a spherical object, we found that it is possible to very efficiently approximate this light point source. This can be done by simply multiply the normalized normal vector of the fragment to be lit, by the light radius of the light source, and add this vector to the position of the light (see Figure 4).

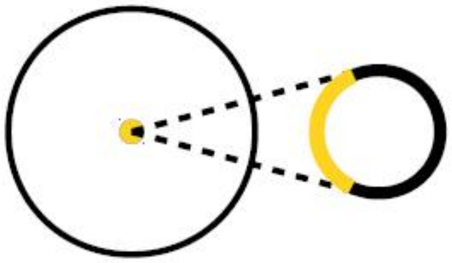


Fig 9: Light represented as a point light source

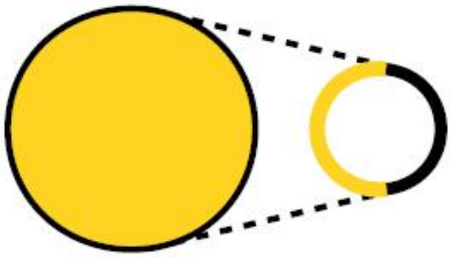


Fig 10: More realistic light, giving light to a larger area of the sphere.

VI.PARTICLE SYSTEM

The effects, such as smoke, fire and explosions, are in general difficult to create due to their irregular, and perceptually random, shapes and behavior of asteroids.

A common way to create convincing effects in real time is to generate and transform them using a particle system, as discovered by Reeves [Reeves 1983]. This section is explaining in the algorithm used in our game, as well as the results, and discusses our decision.

A. Dynamics of Particle System

Each particle has a predetermined lifetime, typically of a few times, during which it can undergo various changes. Dynamic particle system begins its life when it is generated or emitted by its particle system. The system emits particles at random positions within a area of space shaped like a sphere, hemisphere, cone, box or any arbitrary mesh. The particle is displayed until its time is out, at which point it is removed from the system. The system's emission rate indicates how many particles are emitted per second, although the exact times of emission are randomized slightly. The choice of emission rate and average particle lifetime calculate the number of particles in the "stable" state (i.e., where emission and particle expiry are happening at the same rate) and how long the system takes to reach that state.

VII.USER INTERFACE

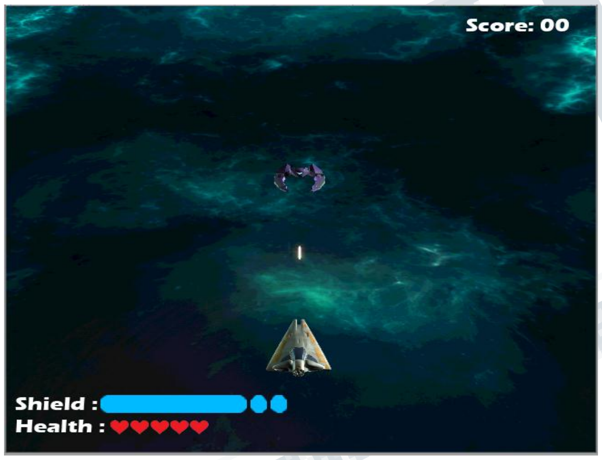


Fig 11: Screenshot during a gameplay

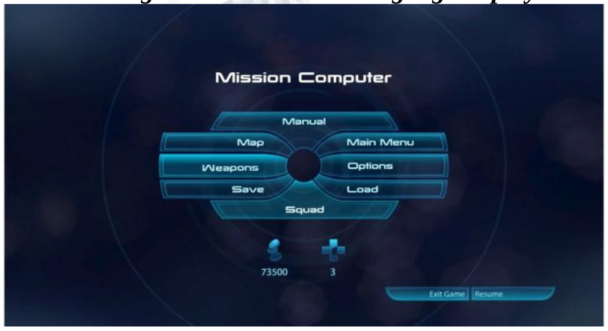


Fig 12: Pause Menu

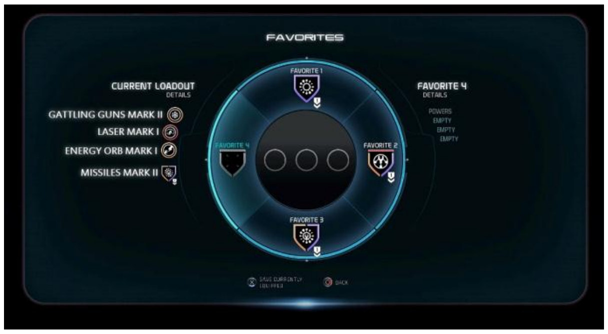


Fig 13: Weapons selection and favorites menu

VIII. TECHNOLOGIES USED

A. C# Language

(For writing object's dynamic behaviours)

C# (pronounced as see sharp) is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

B. Unity3D

(For combining sourcecode with objects)

Unity is a cross-platform game engine developed by Unity Technologies, which is primarily used to develop videogames and simulations for computers, consoles and mobile devices. First announced only for OS X, at Apple's Worldwide Developers Conferencein 2005, it has since been extended to target 27 platforms.

C. Adobe Photoshop

(For designing 2D objects like MainMenu)

Adobe Photoshop is a popular image changing software package. It is widely used by photographers for photo editing (fixing colors, reducing noise, adding effects, fixing brightness/contrast) and by graphic designers and Web designers to create and change images for web pages.

D.Blender (For creating 3D models)

Blender is a professional, free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games. Blender's features include 3D modules, UV unwrapping, texturing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft bodysimulation, sculpting, animating, match moving, camera tracking, rendering, motion graphics, video editing and compositing. It also features an integrated game engine.

IX.PACKAGES INCLUDED

A. UnityEngine.Audio
Loading audio files and playing on triggers, buffering and streaming of audio.
B. UnityEngine.UI
Display of UI and handling user input, drawing of UI, handling the clickable area for buttons and other inputs
C. UnityEngine.AI
Handles the AI movement, player tracking and chasing, auto firing, etc
D. UnityEngine.SceneManagement
Handles different scenes in the game, like home screen, game stage screen, pause screen, gameover screen. SceneManager loads the scenes based on user input via UI or if any event occurs like collision of player vehicle
E. Quaternion.Euler
Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis (in that order). Used to Rotate ships when moving in X axis
F. EventTrigger
Receives events from the EventSystem and calls registered functions for each event.
G. Rigidbody
Adding a Rigidbody component to an object will put its motion under the control of Unity's physics engine. Even without adding any code, a Rigidbody object will be pulled downward by gravity and will react to collisions with incoming objects if the right Collider component is also present.
H. System.Collections
The System.Collections namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries. Required to use classes like IEnumerator.

X.CONCLUSION

Even though the game might not the standards of many commercial games, given the resources and time frame. The game is easy to play and the visual effects make the game look graphically good. Each effect contributes to the appearance of the game. The game is easy to play and the visual effects make the game look best. Each effect donates to the look of the game own laptop. There are a lot of positive aspects to working together with team. When problems occur everyone can help, ideas can be discussed and you get to know each other better, making it more motivating and fun with members to work on a project. In hindsight, we should probably have utilized the rooms with computers provided by college in order to increase productivity.

XI.FUTURE SCOPE

There are some features that had to be dropped during the development of this project due to the lack of time. If the group were to continue further development, some of these features would be reexamined and implemented into the game. More content is something that would be the first thing to be added to the game is that multiple player can play the game at a same time. A proper development system, where the player receives better weapons or more health over time, has also been discussed. This could be implemented in a number of ways, either by collecting elevations dropped by enemies or by receiving an elevation when a certain number of enemies have been defeated by the player. Regardless of the specifics, this will give the player a feeling of development and thus will inspire them to keep on playing. Special powers like time rewind can also be implemented with extended time.

REFERENCES

[1] R. Galantay, et al., "living-room: Interactive, space orientedaugmented reality," 2004, p. 71.

[2] K. Kim, et al., "ARPushPush: Augmented reality gamein indoor environment," 2005

[3] M. WEILGUNY and D. MEDIEN, "Design Aspects in Augmented Reality Games," 2006.

[4] AKENINE-MÖLLER, T., HAINES, E., and HOFFMAN N. 2008. Real-Time Rendering. Third Edition, Boca Raton, CRC Press.

[5] BLINN, J. 1978. Simulation of Wrinkled Surfaces. ACM SIGGRAPH Computer Graphics, vol. 12, no. 3, pp. 286–292.

[6] NYSTROM, R. 2014. Game Programming Patterns. Genever Benning, ch 2.

[7] FISHMAN, Band SCHACHTER, B.1980.Computer display of height fields. Computers& Graphics, vol. 5, no. 2, pp. 53-60.

[8] MUSGRAVE, F. K., KOLB, C. E., and MACE R. S. 1989. The Synthesis and Rendering of Eroded Fractal Terrains. ACM SIGGRAPH Computer Graphics, vol. 23, no.4, July, pp.41-50.