Space - project (tentative)

Game Design Document

Index

Index

Game Design

Summary

Simple side scrolling shooter with different shooting (energy bar) system, halo-style shield/health system, and enemy path mechanics to set it apart. Emphasis will be placed on boss battles.

Gameplay

The goal of the game is to make it from one side of the level (the left) to the other while killing as many enemies as possible, then fight and defeat a boss. The player will control a spaceship that can move anywhere on the screen, and can fire a weapon at incoming enemies. The enemies will come at the player in different ways, slowly getting harder to kill and/or dodge. Boss battle's will be a large part of the game, with difficult patterns of attack and interesting mechanics. The tactics the player will employ throughout the game will be subject to change, depending on the enemy(s)/boss they are fighting.

Mindset

The mindset of the player will mostly be defensive as they go on to harder levels. There will be many things to dodge while also attempting to applying damage to the enemy(s). The player will receive weapon upgrades periodically throughout the game, giving a feeling of 'achievement'.

Supported Platforms

We will start off with win32, then move to android / iOS and other platforms.

Technical

Screens

1. Title Screen
    a. StartGame
    b. Options
    c. Quit
2. Game
    a. Pause
        i. Resume
        ii. Options
        iii. Quit
3. Game Over
    a. Retry
    b. Quit

Controls

Win32

The input will be all from the keyboard, W-S-A-D and the arrow keys for movement, and space bar for shooting.

Mechanics

This is where all of the details of the systems will be described. For implementation details, see Development.

Energy Bar

The energy bar will act similar to the stamina bar from Dark souls. Each offensive action the player makes will consume some portion of the bar. The bar will have a constant regeneration rate, though there will be a slight delay in this immediately after any action taken that consumes energy. When the energy bar is depleted, it must recharge before any energy can be used again. When the energy bar is depleted, the player will only be able to shoot the standard laser gun, at a slower pace. The recharge rate when the bar is depleted is subject to change, though it should be at a slightly slower rate than the standard regeneration rate.

Player Health System

The idea behind the shield/hull system is to be more forgiving for making mistakes (getting hit) infrequently. Should the player get hit by multiple attacks in quick succession, they should likely die.

Shields

The shields of the player's ship will work almost exactly like the shields for players in Halo 3. Each hit the player takes will take a moderate amount away from the shields. After some integer of time the shields will begin to regenerate (quickly) to full. It shouldn't be possible to be 'one shot' at any time if the shields are full, instead if an attack would one shot the ship it will instead completely deplete the shields. There should be a small integer of time (~1/4th of a second) of invulnerability when the shield is broken, ~~unless the shield was taken out in one hit~~. When the shield is broken, the ships hull will be exposed.

NOTE: Maybe have an ability, or upgrade, where the player infuses the shield with the energy bar, effectively making the energy bar a less effective shield bar. Energy bar mechanics should still apply when this happens. This could be effective in defensive positions, but kill any offensive action for a time (the energy bar would be drained.)

Hull

When the ships hull is exposed, it should only take one more hit to blow it up.

Weapon System

The player will have three separate weapon systems. The laser gun, heat seeking missiles, and protective orbiting turret(s). The player will start out with level one laser gun only.

Laser Gun

The laser gun will be the primary weapon. Check Weapon Paths to see how it will look.

Heat seeking Missiles

The heat seeking missiles will curve toward enemies, though should not be able to accurately hit faster enemies. The missiles will fire at a slower rate than the laser gun. As the missiles are upgraded, they will all shot at the same time, targeting different enemies.

Orbiting Turret

Orbiting turrets will be able to shoot in a straight line, as they orbit the ship. When the turret comes into contact with an enemies attack (a bullet) it will cancel it. If a turret makes contact with any enemy, it will destroy the turret, and hurt the enemy it collided with. The

turret's attacks will be weaker than a laser gun, and will shoot at a slower pace.

Upgrades

Each of these weapon systems will upgrade independently of one another. All upgrades should reset at the beginning of every level, but should have the option of starting the player off with a number of upgrades.

Laser Gun Upgrades

The laser gun will have four upgrades. Each level will change the amount of bullets put out by the gun, and the path that they will take. Check Weapon paths to see how the upgrades will look.

Heat Seeking Missile Upgrades

The heat seeking missiles will upgrade three times, each upgrade giving the player one more missile to shoot.

Orbiting Turret Upgrades

The Orbiting turret will have three upgrades. Each upgrade will add an orbiting turret around the player. The first turret should have a fairly wide orbit radius, while orbiting clockwise. The second should have a bit smaller orbit radius, and orbit counter-clockwise. The third will have a still smaller orbit radius and orbit clockwise.

Over-Shield Upgrade

There will be an upgrade that doubles the effective shield amount. When activated, the shield will regenerate (at about the same rate as when energy bar is depleted) to 200% normal shield level by overlaying the shield bar with a different color. The shield should not regenerate when it has >100% remaining. If the player activates the shield upgrade when they have no shields, their shields should regenerate almost instantly, then begin charging the over-shield.

Collision Detection

There will be a z-order for the objects in the game, from the top:

1. Player's projectiles
2. Enemy Projectiles
3. Upgrades
4. Player's ship
5. Enemy ships

We will start with a simple collision detection system with just bounding box detection. From there we will add in pixel perfect collision code into the bounding box detection. Due to the fact that there will be many projectiles on screen, we should probably split the 'collision

map’ up into different sections of the screen, to save CPU time. Collision detection doesn’t need to be checked every frame, we should only check it at some fraction of the current FPS (½, ⅓, ect.).

Level Design

Each level will consist of waves of enemies coming at the player by different paths, and ending in a boss fight. For now we will not have any other objects in levels besides enemies, due to the lack of graphic assets. Each level will be defined in xml files.

Waves

Each wave will have a number of Groups and a time delta which will define when the wave will be initiated. It should be possible to have multiple waves on the screen at once. ~~(this eliminates the need to have smaller time delta's between each group on waves.)~~

Example level file:

```
<?xml version='1.0' encoding='UTF-8'?>
<level name="test-level">
	<wave delta="3000">
		<group type="SimpleEnemy" number="10" delta="300" path="loop1"/>
		<group path="loop2" upgrade="turret-upgrade">
			<enemy type="SimpleEnemy"/>
			<enemy type="MediumEnemy"/>
			<enemy type="SimpleEnemy"/>
			<enemy type="LargeEnemy"/>
			<!-- empty spot in the enemy chain -->
			<enemy/>
		</group>
	</wave>
	<wave delta="4500">
		<group type="MediumEnemy" number="3" path="sine1" upgrade="laser-upgrade"/>
	</wave>

	<!-- there should be a set wait time between the last wave and the boss -->
	<boss type="Boss1">
</level>
```

Enemy Groups

A Group consists of an enemy type(s), the number of enemies (unless there are different enemy types in the Group), the time delay between each enemy appearing (there should be defaults for each enemy type when this is not present), and the path that they will follow. If there are multiple enemy types in a group, then they should be defined explicitly in

the definition file. Individual enemies should have the option of having the 'Upgrade' flag, if they need to drop an upgrade. It should also be possible for an upgrade to drop only after a whole group of enemies is defeated, with the upgrade dropping from the last enemy of the group to be killed. It should not be possible for an enemy within a group of enemies that drop an upgrade, to drop an upgrade, this may cause two upgrades to drop on top of each other.

Example of single type enemy group:

```
<group type="SimpleEnemy" number="10" delta=”350” path="loop1"/>
```

This will spawn a group of ten simple enemies following the 'loop1' path when this groups wave is initiated.

Example of multiple-type enemy group:

```
<group path="loop2" upgrade="turret-upgrade">
        <enemy type="SimpleEnemy"/>
        <enemy type="MediumEnemy"/>
        <enemy type="SimpleEnemy"/>
        <enemy type="LargeEnemy"/>
        <!-- empty spot in the enemy chain -->
        <enemy/>
        <enemy type="SimpleEnemy"/>
</group>
```

This example will spawn a group of enemies in the following order: Simple Enemy, Medium Enemy, Simple Enemy, Large Enemy, empty space in the enemy group, then a Simple Enemy. This group would follow the loop2 path and drop a turret upgrade when the last enemy is defeated.

For a single enemy to drop an upgrade, you must define it as a multiple-type enemy group, even if it is all the same enemy type.

Enemy Paths

An enemy path will consist of a function determining how the enemy(s) will move across the screen. The path's enemies can follow will be decoupled from the enemies themselves so any enemy can use any path. The enemy path's should be independent of screen size (may need to create our own coordinate system to solve this.) We will use a program called Graph to map out functions to use.

Enemies

There will be three base enemy classes, small, medium, and large. We will adjust the hue of the sprites for these three enemies to create more difficult enemies for later stages.

Enemy difficulty will be scaled with frequency of shots, speed along their path, and the frequency of the waves (also, perhaps the number of enemies in each wave.)

Enemy Mechanic Ideas

- An enemy that, when destroyed, breaks into many different smaller enemies and attempt to fly into the player (‘heat seeking’ algo.)

Bosses

Each level will need to end in an interesting boss fight, as that is where most of the ‘fun’ will be. Each boss should have unique mechanics that set them apart. The bosses health should not be shown, but implied by showing damage to their ship with particle effects. Boss encounters should have the ability to spawn waves of enemies. It should also be possible to fight multiples of certain bosses (say, the boss of level five is two level one bosses.)

First Boss

Boss definitions start here.

Boss Mechanic Ideas

- The boss would summon a tunnel made of energy, the player has to navigate through it without touching the sides (one touch kills the shields, next touch kills the player) while the boss shoots at the player.
- A boss with a bunch of ‘drones’, which would harass the player along with the boss. The player would be able to kill the drones.
- A boss that shoots a massive number of bullets, way too many to dodge randomly. We would need to mathematically determine ‘holes’ in the waves of bullets for the player to safely maneuver to/through.
- The boss would have ‘energy bars’ at the top and bottom of the screen. they would periodically switch places by moving toward the player in a ‘heat seeking’ fashion. Bars would slowly get longer and harder to dodge.
- A small, nimble boss that has a somewhat decent shield (similar to the players shield) that can dodge the players bullet’s at a reasonable rate. Maybe put some sort of time limit on this encounter.
- A giant enemy wave would circle the boss. Each wave would be circling in a different direction, making it hard to punch holes without killing them all.

- A harder boss could require the player to have at least some shield by throwing an undodgeable attack at them, leaving them vulnerable, killing them otherwise.
- An attack that boomerangs back to the boss at the other end of the screen.
- an omni-boss, with the ability to do a lot of different bosses maneuvers.
- There are three or so objects on the field. Color coded, numbered, ect. The player has to hit/Destroy them in a certain order to make the boss vulnerable to attack.
- Larger bullets that explode sending smaller bullets in all directions.
- Gravity bullets that cause the player to slowly move toward them. If they touch the center they die.
- The boss can summon 'black holes' on the grid that pull the player in different directions.
- A boss that attacks in a pattern similar to the background music.
- A spinning wheel that speeds up, when it reaches top speed it shoots out a powerful attack in all directions that is hard to dodge. must be hit every so often to slow it down.
- Boss with an external shield that reflects the player's bullets making them hostile to the player.
- An attack that drains the player's energy bar.

Development

The game will be developed using the Entity Component System design pattern with the [entityx](#) framework.

Coding Style

Each file name should be all lowercase. They do not need to match the class name.

Components

SpriteComponent

CCSprite *sprite

This component will represent any sprite on the screen. (note: We should try to package all sprites into one spritesheet so we can use a static SpriteBatchNode to save cpu on opengl calls.) This component will be the main component of entities in the game.

CollisionComponent

CollisionType type

The object type will define the type of collision it should be checked against

Types:

1. Player's projectiles
2. Enemy Projectiles
3. Upgrades
4. Player's ship
5. Enemy ships

PathComponent

std::function path

The path component will be a function that will take one argument, the velocity of the component it will be modifying

PlayerComponent

This will be a tag component to identify what entity represents the player. (there may be a better way to do this in entityx.)

WeaponComponent

ushort laserGun - domain: 0 - 4
ushort missile - domain: 0 - 3
ushort turret - domain: 0 - 3
This component will hold the state of the player's weapon systems.

ShieldComponent

float strength
float maxStrength
float rechargeRate
float rechargeDelay
float inactiveDelay - the delay before shield beings recharging after being depleted.
bool active
This component will encompass the shield/hull mechanics.

EnergyBarComponent

float energyLevel
float rechargeRate
float rechargeDelay
float maxEnergyLevel
float depletedDelay - the delay before the energy bar recharges after being depleted.
Holds the data needed for managing the player's energy bar.

InputComponent

This will be a tag component that will identify which entity will receive player input (this should be able to be put on multiple entities.)

Systems

CollisionSystem

required components: SpriteComponent, CollisionComponent
Sorts through all collidible entities and checks for their collision with other entities depending on their CollisionComponent flag. A player's projectiles should only be checked against enemy ships, enemy projectiles and enemies themselves should be checked against the player and the player's orbiting turrets, upgrades should be checked against the player only.

ShieldSystem

required components: ShieldComponent

Manages the shield for an entitiy. It needs to handle the recharge delay, the recharge itself, and the overshield upgrade. It needs to emit an event at the change of each state in the shield for animation purposes.

EnergyBarSystem

required compnents: EnergyBarComponent

This system handles the depletion, recharge, and recharge delay for the player's weapon system's .

WeaponSystem

required components: WeaponComponent

This system will process the upgrades the player obtains and change the WeaponComponent accordingly. It will also handle setting the player's bullets paths depending on the WeaponComponent's contents when the player attempts to fire their weapon.

InputSystem

required components: InputComponent

This system will process the input from the player and send events out to the entities that have the InputComponent attached to them.

RenderSystem

required components: SpriteComponent

This system will take care of handling adding and removing sprites from the layer in an efficient manner.

BulletSystem

required components: SpriteComponent, Collision component, BulletComponent

Handles creating bullets. Any time any entity in the game wants to fire a bullet, it will emit an event with the information necessary for the bullet system to create and activate the bullet.

What each bullet needs to know:
Types of bullet
magnitude velocity vector of bullet

<u>How a players bullet is fired:</u>
InputSystem raises an event with the entity that is attempting to fire a bullet (some player) to the energy bar system. The EnergyBarSystem checks if the player has enough energy, if so raises an event with the entity as an argument to the WeaponSystem. The weapon system checks to see which weapon is firing, then determines that player's current level in that weapon, then raises the event(s) for those bullet entities to be placed on the screen. The start position of the bullets should be computed inside of WeaponSystem.