2D Top-Down Shooter
for UTM CSCI 352 Spring 2017

Cole Davis and Mel Howard

Abstract

The proposed game will be a multi-level top-down shooter. The player character will use projectiles to defeat increasing hordes of enemies as he or she progresses through the game. The player progresses by defeating a predetermined number of enemies, allowing them to move to the next location.

1. Introduction

The game will let users create a Payer and control it vertically and horizontally using the WASD keyboard keys. The Player will travel to different locations, then fight and kill enemies with projectiles from a ranged weapon. This game's target audience is college aged gamers, and retro game enthusiasts. We hope the target audience will at first be engaged, and then extremely frustrated.

1.1. Background

A top-down shooter is a game where the user controls his or her character from a birds eye view of their avatar and the surroundings. The proposed game is largely inspired by *Journey of the Prairie King*, a mini-game in *Stardew Valley*. *Journey* is a top-down where the player's objective is to survive and defeat waves of enemies in stages, all while collecting items the enemies drop. The character creation and starting menus are inspired by the 2D infinite runner, *Magicite*, which randomly generates the player's character and begins game play immediately afterward.

1.2. Challenges

One of the most daunting challenges of the proposed game is animating the player avatar and enemies in WPF. Another issue we anticipate is updating the player score or "kill count" as the Player shoots at enemies.

2. Scope

When the proposed game is complete, the user will be able to generate a character, move it, and shoot projectiles using keyboard controls. The game will have a minimum of six levels that contain a predetermined and increasing numbers of enemies. When the Player shoots an enemy it will "die," and the player's score will be updated.

In the future, we hope to randomly generate levels and enemies, as well as implement an item drop system with items that power-up the player character. Another goal we have is to enable return fire from the enemies that cause damage to the Player.

2.1. Requirements

These requirements were gathered based on prior gaming experience.

2.1.1. Functional.

- Users will use keyboard controls to move Player around the map
- Player can shoot enemies with weapon – only one projectile can be shot at a time
- Projectiles will either be absorbed by enemies or leave the screen
- The Player must complete their current level to advance to subsequent levels

2.1.2. Non-Functional.

- The (undetermined) minimum frame rate should allow the user to play with suitable performance.
- The average time between a key press and the event it triggers should be 0.5 seconds and never exceed 2 seconds;

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Move Player	User	Med	High
2	Stop Player	User	Med	High
3	Shoot	User	Med	High

TABLE 1. USE CASE TABLE

2.2. Use Cases

Use Case Number: 1
Use Case Name: Move Player
Description: A user playing the game wishes to move the Player away from spawning enemies. They will press an arrow key corresponding with the direction to move in. This will trigger the process to move the player.

1) User decides which direction to move Player
2) User presses arrow key that corresponds with the desired direction
3) Player location is updated as the User holds down the key

Termination Outcome: The Player is now moving.
Use Case Number: 2
Use Case Name: Stop Player
Description: A user playing the game is pressing an arrow key to move the Player and releases the key to stop.

1) User decides where to stop the Playe
2) User releases the arrow key
3) Player stops moving

Termination Outcome: The Player is now halted.
Use Case Number: 3
Use Case Name: Shoot
Description: A user is playing a level in the game and enemies begin to spawn. They will press the space bar. This will trigger the process to shoot projectiles.

1) User identifies enemies spawning
2) User moves Player to aim at enemies and presses the space bar
3) Player shoots a projectile at the enemy

Termination Outcome: The projectile is shot in the direction the Player aimed

2.3. Interface Mockups

These are very rough ideas and are bound the change depending on how progress of the game plays out. The first is the men, where the start button resides, the next is a player creation in which we hope to allow players to be generated before beggining the game.

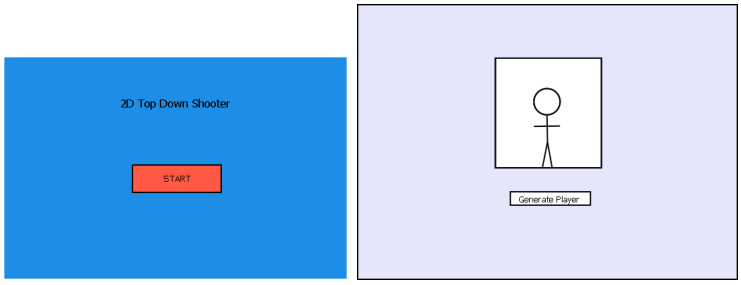


Figure 1.

.

3. Project Timeline

The timeline shows all the major due dates, and whenever there were any large scale updates/breakthorughs made

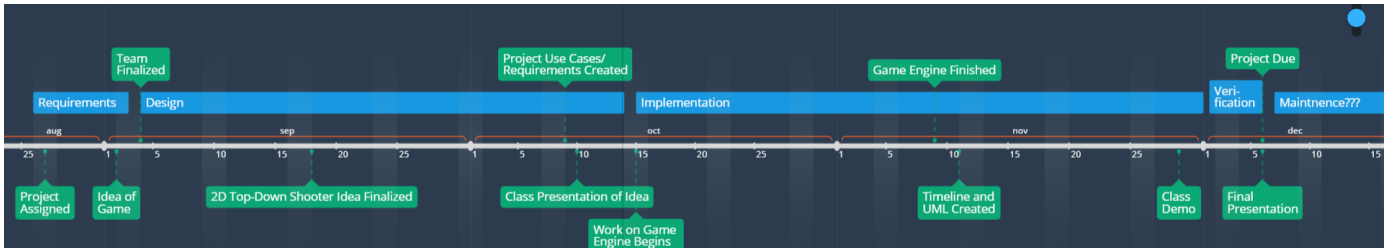


Figure 2.

4. Project Structure

The game, first of all, is split into two namespaces: the Egine and the Game.
The engine is what runs the game, but holds no knowledge of what the will make up the game.
The game, is everything like specific game objects like the player or enemies and determines their logic.

The files included in Engine are: Assets(Directory), GameObject.cs, IPlayAreaControl, Map.cs, PlayArea.cs, Random.cs, and Sprite.cs

Assets: Included in assets is all the images to be used in the game in ”png” file form.
Assets.cs: Searches a whole relative directory and puts all the image paths into an array. This is done so that we can have relative paths while still using uris applied to ImageBrushes that represent the images.
GameObject.cs: Holds all the variables and methods to be used across all objects created in the game whether player, enemy, or bullet. This was done so that those things above can inherit this class use all of the commonalities.
IplayAreaControl: Interface implemented by PlayArea with transformation functions
Map.cs: Is simultaneously a GameObject and an aggregation of GameObjects
PlayArea.cs: Uses the grid ”plane” from InGamePlane.xaml and renders Level which renders gameobjects
Random.cs: Custom RNG
Sprite.cs: Contains a base Abstract Sprite with subclasses Rec, and Circle which

The files included in Game are: AbsAmmo.cs, AbsBullet.cs, AmmoInGame.cs, BulletOnPath.cs, CNormAmmo.cs, CNormBullet.cs, CNormEnemyBullet.cs Enemy.cs, EnemyMoveToPlayer.cs, EnemyMoveToRandom.cs, InGamePlane.xaml, InGamePlane.xaml.cs, Level.cs, Obstacle.cs, Rock.cs, Player.cs, PlayerAmmo.cs, PowerUp.cs, SpeedPowerUp.cs, InvincibilityPowerUp.cs, ScoreKeep.cs, PlayerAmmo.cs, ammo.cs, EnemyAmmo.cs

AbsAmmo.cs: An abstract class that acts as the base for CNormAmmo and CNormEnemyBullet
AbsBullet.cs: Abstract bullet and base class for BulletOnPath and CNormBullet
CNormAmmo.cs: Inherits Ammo and overrides its functions **CNormBullet.cs**: Acts as an implemented template of the ammo obj that enemy and player use
CNormEnemyBullet.cs: Overrides ammo and implements enemy bullets **Enemy.cs**: A child of GameObject.cs. This is a class that defines the basic methods and attributes of an enemy without determening their movement. This way we could have different types that move in different ways, depending on which type of Enemy they are. Each enemy can spawn from four seperate locations **EnemyMoveToPlayer.cs**: These are enemies that slowly move to the player for a randomly generated amount of time, then zip towards the player by a dispatch timer at a different randomly chosen time. The reason that they all have random movement times is because if they all had the same, they tended to clump up on top of each other quickly. They move by dispatch timer.
EnemyMoveToRandom: These enemies dont really move to completely random places. They just move to the top of the screen then move randomly while shooting down projectiles towards the player.
InGamePlane.xaml: The Window in the main window
InGamePlane.xaml.cs: Draws the PlayArea in the window
Level.cs: Puts everything together in context

Wall.cs: A child of GameObject.cs, this is how boundaries for the world are set where players can not walk.
Player.cs: The player class that is a child of GameObject.cs that lays down all the logic for player movement, hit detection, and shooting. All the controls are determined in Player.cs. This makes it easy to update with the player. All functions that the player can do are in this class. Bullets are shot up using a dispatch timer.
ammo.cs: A GameObject that sets the attributes for all ammo.
PlayerAmmo.cs: Child of ammo that gives the logic for player ammo, determening which way to be shot depending on which direction string is true on that update cycle. The directio string is gotten from the player class and fed into ammo by a function to set it.
EnemyAmmo.cs: Also a child of ammo that determines the logic for enemy bullets. They just shoot downwards, as the shooting enemies always go to the top of the map to shoot.
PowerUp.cs: A child of GameObject that sets the base for all other powerups.
SpeedPowerUp.cs: If touched by the player, the players speed is increased by use of a paramter in the constructor. The time that the speed lasts is determined by a certain amount of ”frames” or game cycles.
InvincibilityPowerUp.cs: Similar to SpeedPowerUp except it instead sets the player to be invincible and the player sprite flashes.
ScoreKeep.cs: Responsible for keeping track of score and player health

4.1. UML Outline

This UMl was generated using a program built in to Visual Studios showing all the classes inheritence. We had to move around all the classes and such though as what is generated by default is a mess. It does not show dependencies so another porgram called Resharper was used to create the dependency graph below.

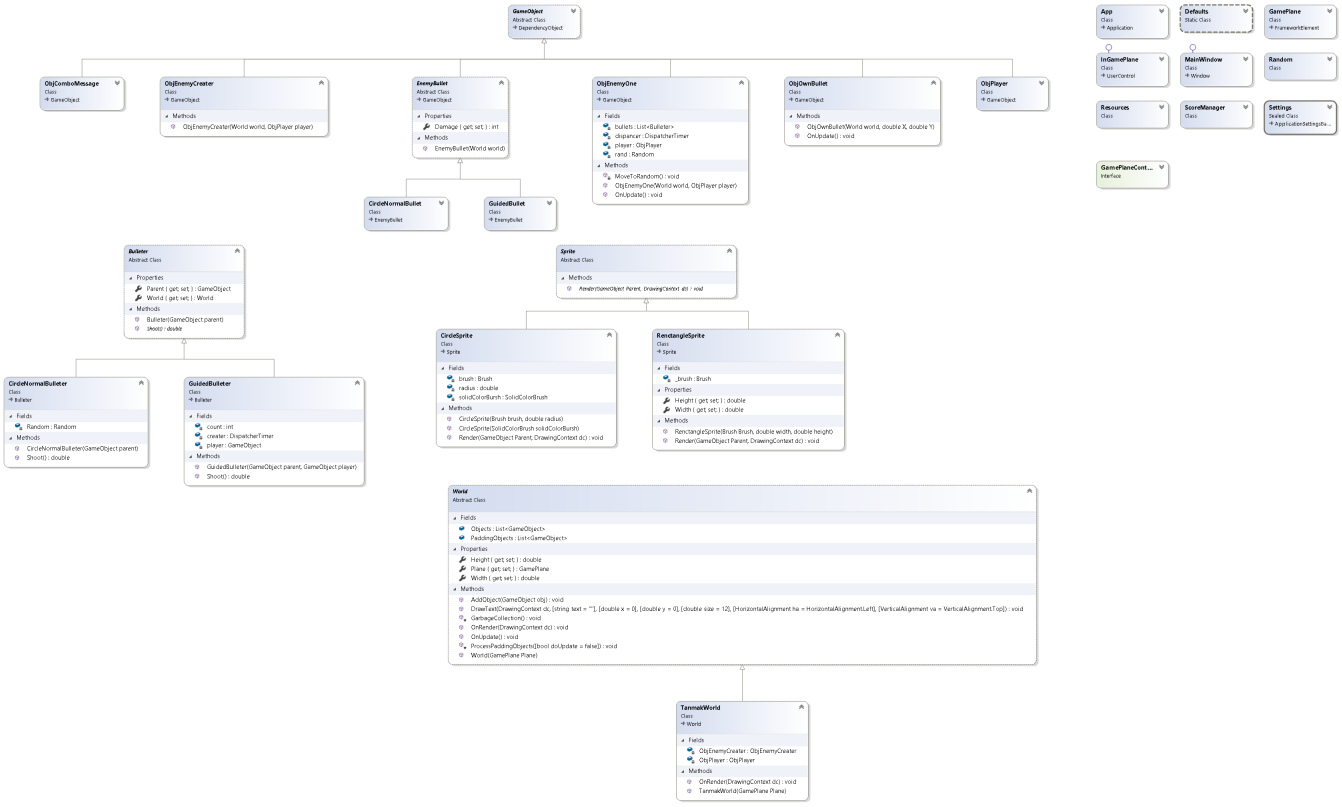
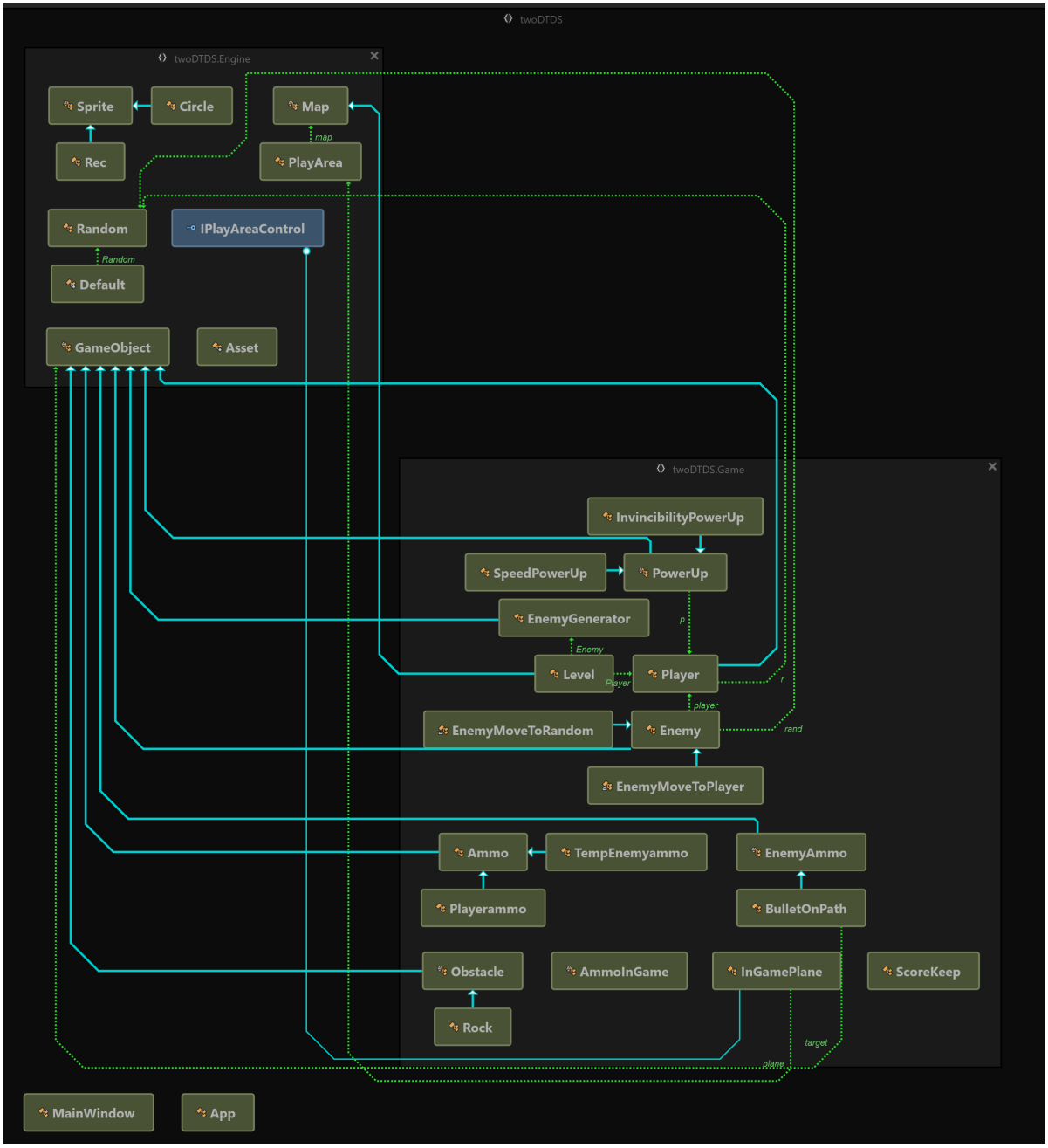


Figure 3.



Graph.png

Figure 4.

4.2. Design Patterns Used

First Design Patter: **Template Pattern**
How this works is that we have a single enemy class that contains all of the base logic for the enemy, Enemy.cs, which has functions likes its OnUpdate(), a spawner function, and their hit detection. Then there are two child classes: EnemyMoveToPlayer.cs and EnemyMoveToRandom.cs. Each of these have a different algorithm in which enemy movement and behavior is used, while all the rest of the enemy logic in Enemy.cs stays the same.

Second Design Pattern: **Observer Pattern**
How this works is that we have a base class for all objects that are part of the game called GameObject.cs. Then, inside our class Map.cs, we delcare a list of GameObjects that all of the objects are added to. Inside map, there is a function for

whenever the game updates, and a change is made, all the objects inside the list of GameObjects are updated using the OnUpdate() method. That way every time there is a change in anything. every game objedct gets updated accordingly.

5. Results

Deliverable 1:. The team is Mel Howard and Cole Davis. The idea so far is to have a game of some sort. There are no final concepts.

Deliverable 2:. : The idea for the game was decided upon to be a 2D Top-Down Shooter. We have a fairly solid idea of how we want it to play and we know our goals. So far, all we have is a square that can move around by using the ”WASD” keys, which is a decent start.

Deliverable 3:. We made some use cases to define some definite things that we want the user to be able to do, like move with arrow keys (which we have already acheived), and shoot (not yet acheived). We have somewhat of an idea of how the game is going to look. We hope to implement a menu where you can roll for stats and generates a chracter, and then move into the game itself.

Deliverable 4:. At this point we have made a huge leap in development, with a game engine being fully implemented inside its own engine namespace, along with a few classes created in a game namespace. Some of the most important classes we have implemented (if not fully, almost fully) are an abstract game object, a map/level, a play area, an array of various ammo types, a player, and an enemy. You can shoot bullets, and so can the enemy. No sprites have yet to be added, so right now its just windows shapes. There is also no hit detection yet, and the bullets can only go in one direction.

Deliverable 5:. Sprites have been added along with backgrounds, but they are not yet the final art we hope to use. The player can also now shoot in multiple directions. There are still no hit boxes, and ammo and enemy logic is still to be created. The idea for chracter/stat generation is most likely dead.

Deliverable 6:. A lot has been added by this point. Player can now get hit by enemy bullets or enimies, there are two enemy types that spawn, powerups have a chance at dropping, sprites have been refined, enemies can be killed by player shooting at them, you can ”roll” to avoid being hit by enemies, there is a camera shake when player gets hit along with invincibility frames, and player can die when health drops to zero.

5.1. Future Work

I definitely want to polish this (music, sprites, hitboxes) and post it on Itch.io as well as potentially writing a tutorial in the form of a post on portfolio blog. Cole and I have also discussed ”porting” it to a game engine like Unity or Unreal and seeing what we could accomplish there. Personally, I want to get a level generator working.