Team CodeCrusaders presents...

RoadRage

A game by Manjesh Puram, Connor Strom, Emma Fletcher, and Logan Kloft

RoadRage is inspired by the idea of a car chase. By the end of our project you will observe rampaging motorcycles, an explosive tank, and a mighty helicopter among other enemies trying to stop our getaway vehicle. Will you be able to make it out alive, or will you be consumed by the *road rage*?

I. Introduction

The project vision document is designed to outline the basic behaviors we want the player and enemies to exhibit, and when they should exhibit them in the case of an enemy. The goal of creating a project vision is so that we can stick to the original design specifications that we came up with, and to provide a starting point for outside viewers to understand what we intend to accomplish and to hold us to this vision. That being said, values such as health and points are open to change throughout the design process since it is difficult at this time to determine how fast the player will shoot, how much damage the player will deal, etc.

At the end of the document you will notice a few extra sections, these are not necessarily part of the visual design you will be experiencing, so much as implementation details. The details will change as we improve our design choices through iterative additions to the code base.

II. Player Behavior

The player currently has three ways to move. Using ‘wasd’, the arrow keys, or a controller joystick to move forwards, backwards, left, and right. In the future we plan to have an option in the main menu, and maybe even during gameplay to map the movement keys to something different. The user may either hold down the left shift or right shift key to enter “slow” mode. Once in slow mode, the user can dodge tightly clumped shots with more success. The player is limited in their range of motion by the boundary of the viewport, which is the visible area you see on the screen. Due to the fact that the developers have different resolution monitors, we decided to limit the viewport to a constant width and height. In the future we might explore resizing the viewport and working with relative distances. However, for now we don’t want to introduce that added complexity. In monogame there is not a circle object, which makes creating a circle hitbox difficult for the player. Thus, we are planning to use a rectangle hitbox which will make collision detection simpler. The player will have three lives since any more might make the game too easy, especially considering that the current shot patterns are not that complex. When we start implementing the more complex shot patterns, we can balance the lives through dropped power ups if need be. While on the topic of powerups, we believe that the simplest powerups involve changing the basic shot pattern such that the player shoots more frequently or generates more shots at once. Thus, we will start at those two modifications for the powerups and might expand from there. The current sprite sheet of the player is not final, but likely to be what we use. The benefit of the current sprite sheet is that it contains four different colored variations of the car. This means that we can add a feature to the menu that allows the player to select their car color. The player follows the singleton pattern since there is only ever one player and we need access to it in multiple places. In the future, if we find that we are able to easily pass around the player reference without cluttering function arguments, then we'll consider moving away from the singleton pattern to encourage a more well thought out design.

III. Phases & Enemy Behavior

The following are the time and content breakdown of the main game. We are following the pattern laid out in the requirements: a wave of grunts followed by the mid boss fight followed by another wave of grunts before ending with the end boss fight. These are the details we aim to work towards for our final product.

Main Game (0:00 - 3:08):
Character(s): Player, Motorcycle Grunts, Police Grunts, Mid Boss, End Boss
Description: The main game lasts for 3 minutes and 8 seconds. It consists of four different phases as briefly described above. The characters that we have planned to be involved in the different phases are motorcycles and police cars for the grunts, a tank for the mid boss, and a helicopter for the final boss. Meanwhile, the player currently takes the form of a fictional red car. However, the sprite sheet that the car is loaded from contains three other color variations which presents an opportunity in the future to allow the player to change the color of the car while keeping its behavior the same. In the case that we find more appropriate assets or add extra enemies, then we will update the current sprites. This might have an impact on hitboxes in the future if we also scale the enemies differently based on their sprite. Although we want to adhere as best as possible to the document, we also want the game to be enjoyable which might require changing some of the following design decisions. In the instance of these changes, we will provide a reason as to the change since we want to make only necessary changes that will improve quality.

Grunts Phase 1 (0:00 - 0:48):
Character(s): Player, Motorcycle Grunts

Description: The motorcycle grunt is designed to spawn in both corners of the top half of the screen. Two are spawned at a time, and they are designed to oscillate left and right (using the LeftMovementPattern and RightMovementPattern classes), meeting in the middle and then driving back to their respective corner. They fire at the player in a half-circle pattern either until they are defeated, or a boss fight is triggered. This enemy will have a low health pool since we may end up spawning more than two at a time. The player will receive 500 points for defeating one of these enemies. Shot pattern of the enemy is subject to change. Enemy can also be used as a support unit for bosses during boss fights

Mid Fight (0:48 - 1:18):
Character(s): Player, Mid Boss
Description: The Mid Boss is designed to spawn at the top left of the screen, and move in a triangular pattern around the top half of the screen (using the TriangularMovementPattern class). During the movement pattern the Mid Boss pauses in each corner for roughly two seconds before continuing on to the next point in the triangle. While moving through the triangle it fires one shot pattern every two seconds. The Mid Boss currently uses two different shot patterns: a Half-Circle shot pattern which fires eight bullets and a Full-Circle shot pattern which uses sixteen bullets (using the HalfCircleShotPattern and CircleShotPattern classes). As the Half-Circle shot pattern fires fewer bullets, these bullets are larger in size compared to the bullets fired with the full circle shot pattern. Currently, the Mid Boss fires 3 Half-Circle shot patterns before switching to 3 Full-Circle shot patterns, and so on. The point values associated with the Mid Boss are not guaranteed yet, but currently, the plan is to award the player 5,000 points if the Mid Boss is killed.

Grunts Phase 2 (1:18 - 2:08)
Character(s): Player, Police Grunts
Description: The grunts phase 2 uses police car sprites as the enemies. They move from left to right at the top of the viewport similar to the movement pattern in phase 1. Police car grunts don't cross the middle when moving left and right. The shot pattern of the police grunts is a constant line of shots with a small gap between each shot that aims towards the bottom center of the viewport. Their health values will probably be similar to that of the motorcycle grunts in phase 1. Currently the grunts aren't very numerous, so we are also considering how we might make the fight more interesting such as adding more grunts that enter and exit the viewport. Or perhaps the police cars try to ram the player and if they miss, exit the viewport. The points for killing a police grunt is the same as killing a motorcycle grunt, 500.

End Fight (2:08 - 3:08)
The final boss fight is estimated to last 1 minute. The boss will switch stages depending on how much time has passed and how much damage it has taken. Although the exact health value isn't known right now, we plan that each stage will be allotted 25% of the boss's health. If the boss is still in stage 1 and drops to 75% health, it will preemptively swap to the next stage. Alternatively, if the boss has above 75% health but 15 seconds have gone by, the boss will swap to the next stage. Now, if the boss is in the second stage and it drops to 75% health, it will not change to the next stage. Instead, the player must damage the boss to 50% for it to change to the next stage or wait for 15 seconds. The player will gain points for every 25% damage it deals to the bosses health in addition to an extra boost of points if the boss is killed. If the player forces the boss to transition to the next stage before its 15 second timer, then the player will be rewarded bonus points based on the time left in that stage. When the player forces the boss to switch to the next stage, the timer will restart at 15 seconds rather than adding the remaining time to the next timer.

There are many additional ways to reward extra points to the player such as a flawless boss run (the player does not take any damage). The point values are not yet set in stone. But for every 25% health the boss loses, 2500 points should be rewarded. And if the boss is killed, an extra 10,000 points. For every spare second the player has in a stage, they will be rewarded 1000 points. This value might be the most prone to change depending on how long it takes for a player to complete a particular stage of the boss.

There are a few ways to display the boss's health. We might have one health bar for all stages, or separate health bars for each stage. At this point, having a single health bar for all stages makes the most sense since the player is rewarded for damaging the boss at particular health percentages.

End Fight Pt1 (2:08 - 2:23):
Character(s): Player, End Boss
Description: The first part of the end boss will mimic the final boss in the demonstration video during the timeframe 2:08 - 2:23. The sprites won't be mimicked, however, one can expect similar movement patterns and shot patterns as in the video. An important differentiation between the two will be the length of this part of the fight. The video's fight lasted for about 45 seconds, while ours will only last 15 seconds due to the imposed total time constraint of about 3 minutes.

End Fight Pt2 (2:23 - 2:38):
Character(s): Player, End Boss
Description: The second part of the end boss fight will mimic the final boss in the demonstration video during the timeframe 2:23 - 2:38. The sprites won't be mimicked, however, one can expect similar movement patterns and shot patterns as in the video. An important differentiation between the two will be the length of this part of the fight. The video's fight lasted for about 45 seconds, while ours will only last 15 seconds due to the imposed time constraint of about 3 minutes.

End Fight Pt3 (2:38 - 2:53):
Character(s): Player, End Boss
Description: The third part of the end boss fight will start with the boss positioning itself in the upper center of the screen. The boss will remain stationary for the entire third phase in this position. Then the boss will fire targeting shots that follow the player for a duration of time before colliding with the player or disappearing. In addition to targeting shots, the boss will fire off a cone of shots that alternate back and forth between two different rotations so that the player must weave between the cone-shaped pattern of shots while at the same time avoiding the tracking shots.

End Fight Pt 4 (2:53 - 3:08):
Character(s): Player, End Boss
Description: During the final stage of the boss fight the boss will move from right to left at the top of the screen. It will initially start from the center, which is the same position that the boss is in during the third phase. The shot patterns will consist of shots moving from either the left side of the screen to the right or from the right side of the screen to the left. Between these thick walls of shots will be a narrow gap for the player to move in between. The player must either survive this phase or beat the boss in order to win the game.

IV. Team Development Roles For Milestone 1

For this milestone, the following people were responsible for these aspects:
Manjesh: Player
Connor: Grunt Enemies
Emma: Mid Boss
Logan: Final Boss

V. Future Design Considerations

The following are a list of some of the future design considerations we will have to tackle. The team is looking forward to implementing these features to make the game the best it can be. The features should be both well designed and feel the right way when put into the game. Since these considerations are out of the scope of the first milestone, we have not deeply considered how we should design them. Nonetheless, we would like to at least propose our initial thoughts in order to start a basis that supports well-informed and intentional designs.

Hit Detection
Sprites are used to represent players, enemies, and bullets. The most important part of the sprite is a rectangle that determines where on screen the sprite is displayed. We can use a separate rectangle that is a part of the sprite class to be our hitbox. A design pattern that would support a hit detection system is the command pattern. One potential architecture is having the ShotPattern classes be invokers for a Bullet which implements the command interface, and then the player can be the receiver that simply damages itself.

Support System
The support system considers powerups, bombs, and rewards. We are asked to implement at least one of them. We figure that a reward and powerup go hand in hand. It makes sense to reward players for killing particular enemies, so why not make the reward a power up or bomb that is dropped upon killing certain enemies? During the discussion of the player, we stated that power ups will most likely boost the player's attack power. Meanwhile, bombs could interrupt the bullets on screen, granting the player a short recess from the onslaught of projectiles. This might be accomplished through an observer design pattern that all ShotPatterns are subscribed to so they can send the Shots for garbage collection.

Powerups
To expand on the two ideas presented in the player: faster shooting speed and more bullets spawned at once. We might have simple power ups that double the bullet speed or triple the bullet speed. Then for the shots, we can similarly double and triple the amount of shots and spread them in a cone shape. We might design this by having the player shots be based on a count and speed factor, and changing one or both affect the bullet output.

Bombs

Bombs can be displayed above or below the lives count and we can make it so they stack to three and reward extra points for collecting bombs while at the max. To use a bomb, it makes sense to let the player choose when to use the bomb, unlike a powerup where we want them to be instantly used upon consumption. Thus, we can have the ‘b’ key be used for a bomb. A bomb’s behavior is such that it clears all enemy shots on the screen. We might implement this through the observer pattern and Shots can store a visible variable that controls whether the shot is both visible and damageable or not. This is so that while we are waiting for garbage collection, the shots are instantly taken away from the screen without any sort of unexpected consequences.

Reward (life, bomb, etc)
Rewards are either powerups, a life, or bomb that is dropped when an enemy is killed. It makes sense to drop a life or bomb from tougher enemies like bosses and power ups from grunts in order to clear them faster. The reward can resemble a care package on the ground with a symbol of a heart, bomb, or 2x, 3x with an image of shots or fast forward arrows for the respective support item. The behavior of collecting a reward is specific to each support item.

Life System
A player will have three extra lives and can only gain them from rewards dropped by specific enemies. Once the player runs out of all three lives, and then dies the final time, the game will end since the player has no more lives left to use. The life system need not be complex, but it should have the ability to end the game and send the player back to the main menu. This means that the system we design must account for the fact that it needs to resume its initial game state. We might accomplish this through the memento pattern.

Menu & Input Mapping
The menu is a resting place for the player before starting the game. In the menu the player will have the option to remap their input keys so that way they can use a configuration that best supports their play style. In class we talked about the command design pattern being a good solution for GUI elements. We will consider this in our menu implementation.

Level Interpreter
The level interpreter is quite a ways off, however, it makes sense to use the builder pattern with either a Factory or Abstract Factory pattern to create and assemble enemies into a level that can be consumed by the game. The product should only be returned

when Game1 wants to start the level, or after the player decides to continue from the menu, thus completing the use cases of a builder pattern. If we want, we can also support a simple level creator that creates the documents of a level rather than interprets them. This will make creating levels from scratch easier, but is out of the scope of the project.

Score System
The scoring system needs to update whenever a player scores points. The best way to do this might be having global access to the scoring system which means that we can use a singleton pattern for the scoring system. Using a singleton pattern means we won't have to pass around the scoring system to the entities that can reward points. At this point, the display of the score is not finalized, and neither are that of the bombs and lives for that matter. It might be worth having them near the top of the screen and overlaid over the background instead of creating a separate panel for them that shrinks the playable area.

Cheating Mode
Cheating mode involves creating a way for the player to not take any damage given that they collide with an enemy or shot. The most obvious way of accomplishing this is by adding a boolean member to the Player that signifies whether the player can be damaged and checking the boolean whenever a hit is detected. The cheating mode can be toggled on and off using the 'c' key for cheat mode. We also need a way for the player to remain invulnerable for a few seconds after getting hit, cheating mode has the potential to also allow for this so that way we don't have to create two different systems for two very similar use cases.

Difficulty Settings
This is an optional setting specified in the project specification document. There are a variety of ways of increasing difficulty such as increased speed, increased health or lower damage, less lives, more enemies, more enemy bullets. The most straightforward of these solutions are probably changing health, damage, and lives values since the changes required are more likely to be trivial. Whereas changing speed or enemy and bullet behavior might not be as trivial. We will have to consider how we can make our design flexible so that it can potentially edit enemy count and shot patterns. Changing the speed might be possible if we define everything relative to a single speed variable. This solution seems like it has drawbacks because we won't have the ability to predict the changes to every feature that depends on speed, in other words using a single

speed value as a basis for everything and changing it could have unintended consequences.

Secret Requirements
By ensuring that we have flexible designs we will be able to incorporate the secret requirements that Professor Zeng plans to introduce later in the semester.

This marks the end of the project vision document. Thank you for viewing our initial design plan.

Documentation

Basic Class Information

Sprite Class

- DestinationRectangle is the bounds of the sprite on the screen
- It is also the hitbox - Circle hitbox might be difficult since it's not supported directly by MonoGame's API

Background Class

- To add a background:
    - Create a sprite
    - Call the sprite's LoadContent method to set its display image
    - Call background's AddBackground method

Player Class:

- Singleton
- Accessible anywhere in the project

Shot Class:

- Defines its own movement

ShotPattern

- Instantiates and Controls shots
    - Rotation, speed, direction, amount, etc.

MovementPattern

- Requires access to DestinationRectangle of a Sprite
- Defines the movement of the Sprite

How to Create a New Enemy:

The basic enemy class comes equipped with a MovementPattern, ShotPattern, and Sprite. To create an enemy simply create a new instance of one:

```
Enemy grunt = new Enemy();
```

Add a Sprite

Now, to add a Sprite to the enemy, you must call the enemies LoadContent method:

```
grunt.LoadContent(Content, “Grunt”);
```

Add Movement

To add movement behavior to the enemy, add a MovementPattern:

```
MovementPattern mp = new CircleMovementPattern();
grunt.MovementPattern = mp;
```

Create Shots

When you want the enemy to shoot add a ShotPattern:

```
CircleShotPattern csp = new CircleShotPattern(16);
csp.CreateShots(Content, “01”, grunt.Center);
grunt.ShotPatterns.Enqueue(csp);
```

Called Every Frame

For the MovementPattern and ShotPattern to work properly, we must call the Update and Draw method of Enemy every frame. This can be done in Game1.cs or any other Update or Draw method that is called every frame.

Then in Update() of Game1 add:

```
grunt.Update(gameTime);
```

Then in Draw() of Game1 add:

```
grunt.Draw(_spriteBatch, gameTime);
```

How to Create a Movement Pattern

Inherit from the base MovementPattern class. Then override the Move() method. This method has access to an Enemy reference called enemy. To move the enemy, manipulate the enemies DestinationRectangle. The DestinationRectangle is a box where the sprite is displayed. Making the DestinationRectangle larger will make the sprite larger, changing the x position will move it accordingly. The DestinationRectangle is a member attribute of the Sprite class.

How to Create a ShotPattern

Inherit from the base ShotPattern class. The ShotPattern class has an Update(), Draw(), and Finished() method. In the Update() method you should call the Update() method of any Shots contained in the ShotPattern. In the Draw() method you should call the Draw() method of any Shots contained in the ShotPattern. The Finished() method should return true when the ShotPattern is not useful anymore, i.e. all Shots are offscreen or have collided with the player. The Enemy will call this function so it knows when to Dequeue the ShotPattern.

How to Create a Shot

Inherit from the base Shot class. The base shot class is a Sprite, so you can set the sprite of an instance by:

instance.LoadContent(Content, “FileName”);

You must implement the Move() method in the derived class. In the Move() method you will define how the bullet moves across the screen. Manipulate the base.DestinationRectangle to make the Shot move.

The Shot class also provides the Collided() and Offscreen() methods which can be left as default assuming the hitbox is the same as base.DestinationRectangle. These methods will trigger an event that can subscribed to whenever the bullet is observed in either the collided or offscreen state. These events will only trigger once unless one manually sets _offscreen or _collided to false. Finally, the Shot class includes an InvokeBulletEnd() method which triggers an event that means the Bullet is not useful anymore. This event is triggered by the Offscreen() and Collide() event, therefore it can be triggered more than once.