Date: 03/23/2020

Team 7

Iteration 2

Members : Paul Patraca Pantoja
Utsav Dhungana
Bipul Karki

“Carronade"

Project Plan:

The game has had more features introduced in this iteration. The cannon is able to attack the player with different weapons and give powerups. The enemy, the cannon, is controlled by the computer AI while the player is controlled by the user. Players will be controlled by the user via the keyboard. A health bar has been introduced in this iteration along with the enemies shown. All the planned requirements for this iteration have been met.

The game is written in C# using framework software Monogame that runs on the .NET 4.6.1 framework, and is compiled with Microsoft Visual Studio.

The most closely related games are similar to marathon games, such as Jetpack Joyride or Subway Surfers. How does “carronade” differ from these games? These games do not have a health bar which decreases overtime. Instead in these games the game is over when the player character hits the obstacle/enemy. They are not minimalistic which takes the smoothness of the game away.

Iteration 1

- Content Management
- Test cases created for each feature implemented
- Basic Player Locomotion
- Basic Enemy Functionality

Iteration 2

- Player able to boost character’s life with power-ups from cannon
- Projectiles with greater destruction ability
- Score tracking implemented
- Updated game based on previous feedback
- Test cases for all new features introduced
- Getting feedback from users(our colleagues) and fixing bugs.

Iteration 3

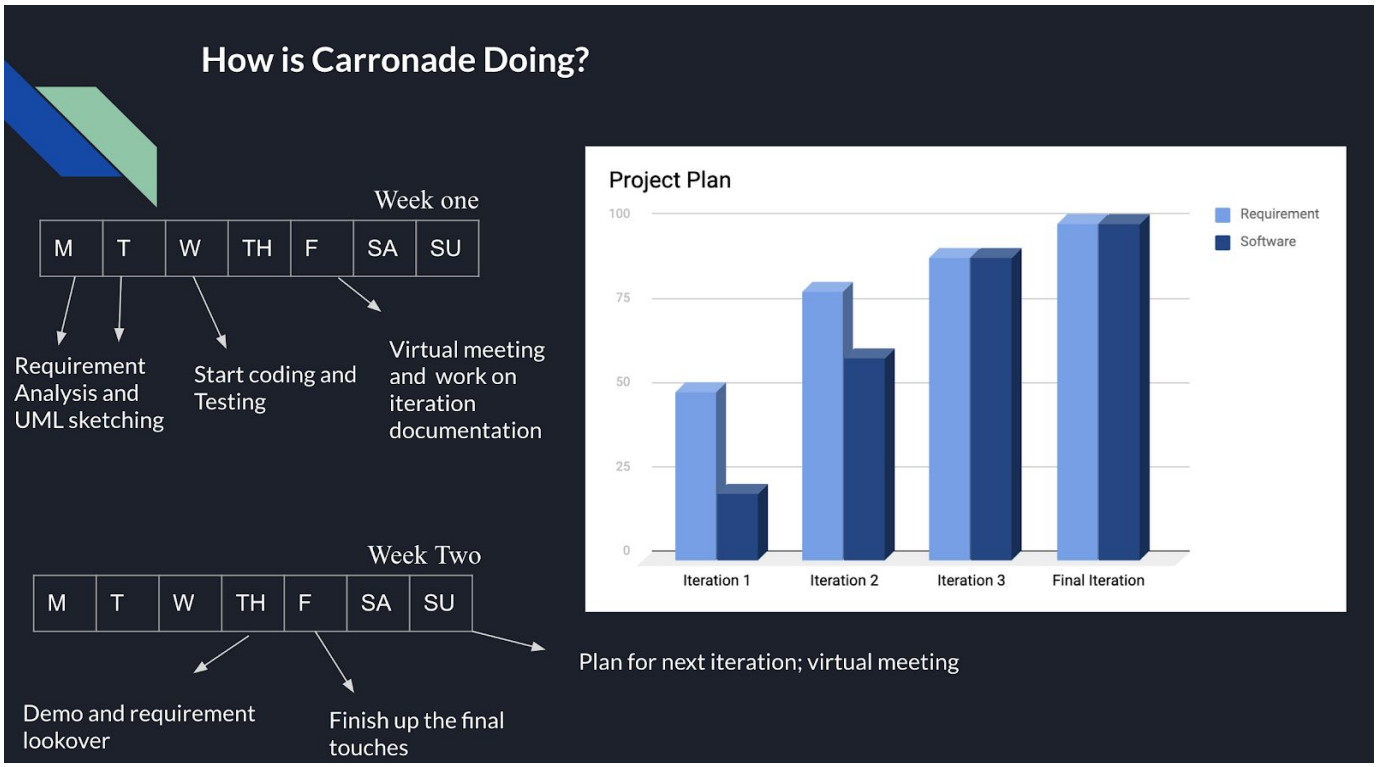
- Main Menu, Settings Menu, Selection Menu

- Able to select new different characters, cannons, and arenas
- Sharpening gameplay smoothness
- Fixing bugs and faults based on previous feedback

Final Iteration:

- Finalizing the graphical and animation implementation
- Fixing bugs and faults based on the feedbacks from users
- Creating final test cases for all functionality

Drop-down of our Project Plan:



Top Five Risks:

1. Changes in the functional requirements.
    - Probability of the risk is 50%, and the effect of the risk is we will have to spend 20 more hours. So, the Risk Exposure (RE) = 0.5 * 20 = 10 hours.
    - To mitigate this risk we will test first and implement the game functionality.
    - Changes in the game functionality might be made during each iteration.

2. Unidentified Risk : Includes a member getting sick.
    - Probability of the risk is 40%, and the effect of the risk is we will have to spend 20 more hours. So, the Risk Exposure (RE) = 8 hours.
    - Creating a healthy living environment and good hygiene would mitigate the risk.
3. Poor programming practices leading to memory leaks and slowing down the user's machine; occurance of bugs.
    - Probability of the risk is 40%, and the effect of the risk we will have to spend 10 more hours. So, the Risk Exposure (RE) = 0.4 * 10 = 4 hours.
    - Inexperience in loading in assets to memory alongside time constraints make this the most possible scenario.
    - Careful and thoughtful use of documentation and available resources can mitigate the probability of the risk during each iteration.
4. Exceeding the deadline to complete the project i.e project not completed on time.
    - Probability of the risk is 20%, and the effect of the risk we will have to spend 20 more hours. So, the Risk Exposure (RE) = 4 hours.
    - Hard to gauge at the moment, but the most common project flaw. Scope creep and an inability to pace can lead to this. Being mindful of capabilities and time remaining can negate this risk.
    - Making a good timeline for each iteration would solve the problem.
5. Poor communication between the team members.
    - Probability of the risk is 20%, and the effect of the risk is we will have to spend 15 more hours. So, the Risk Exposure (RE) = 3 hours.
    - Strong communication between the members throughout the class project using mediums like GroupMe would mitigate the risk.

Inputs and Outputs:

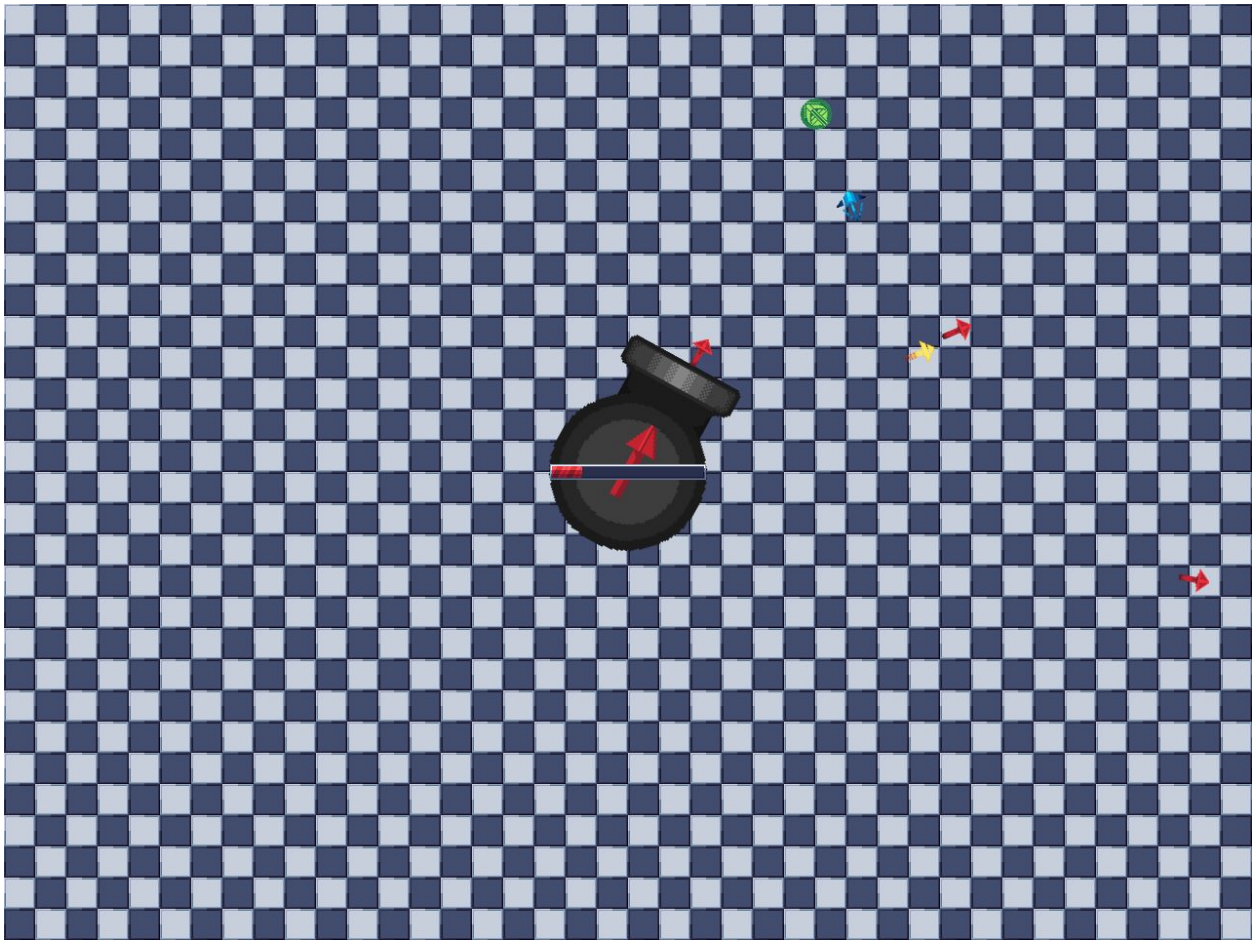
Input	**Output**
Keyboard Button (S)	Player moves down the screen
Keyboard Button (A)	Player moves towards left
Keyboard Button (W)	Player moves in upward direction
Keyboard Button (D)	Player moves towards right

Keyboard Button (R)	Resets the game and switches the player type
Keyboard Button (Shift)	Players go faster for the first or teleport for the second.

Data Structure Implementation:

```
//It's easier to define how to handle each asset inside the asset definition itself.
public partial class XMLAssetBuilder {
    public Animations BuildAnimations(System.Xml.XmlNode node) {
        Texture2D tex;
        int id, w, h;
        string defaultAnim;
        //TODO: Validate XML. Because there is absolutely 0 XML format validation going on here.
        try {
            tex = contentManager.Load<Texture2D>(node.ChildNodes[0].InnerText);
            id = int.Parse(node.ChildNodes[1].InnerText);
            w = int.Parse(node.ChildNodes[2].InnerText);
            h = int.Parse(node.ChildNodes[3].InnerText);
            defaultAnim = node.ChildNodes[4].InnerText;
        } catch(Exception e) {
            Console.WriteLine(e.ToString());
            return null;
        }
        Animations anim = new Animations(tex, id, w, h, defaultAnim);
        Console.WriteLine(node.ChildNodes[5].InnerText);
        foreach (System.Xml.XmlNode animation in node.ChildNodes[5].ChildNodes) {
            Console.WriteLine(animation);
            string name = animation.ChildNodes[0].InnerText;
            int s = int.Parse(animation.ChildNodes[1].InnerText);
            int e = int.Parse(animation.ChildNodes[2].InnerText);
            anim.GenerateAnim(name, s, e);
        }
        return anim;
    }
}
```

A prototype of the game



Here, the green circle is the player, and the red, blue, and yellow arrows are the enemies shot by the cannon in black (the enemy director). The score will increase for the player depending on how long they survive for.

Requirements:

Req. ID	Requirements	Functional / Non-Functional	Implemented in Iteration	Updated/ Deleted/ Added Req. to Document	Priority
R1	The enemy shall throw objects towards the player when game starts	Functional	1		Does
R2	The System shall display health for the convenience of the Player	Functional	2	Updated	Does
R3	The Player shall be able to mute the sound of the game.	Functional	3		Could
R4	The System shall not be error prone in a manner that will inconvenience the user	Functional	1		Must
R5	The Enemy director shall create at least two boosts for the Player	Functional	2	Updated	Does
R6	The Player shall be able to choose from at least two different characters.	Functional	2		Does
R7	The Player shall be able to choose from at least three different arenas.	Functional	3		Could
R8	There shall be tracking of scores so the player can make a note of their own progress.	Functional	3		Must
R9	The Enemy director shall create at least three different types of enemies.	Functional	2		Does

R10	The Player shall be able to change the graphical settings of the game.	Functional	3		Could
R11	The player's avatar shall move responsively to the inputs of the player.	Functional	1		Does
R12	The player shall be able to restart the game.	Functional	2		Does
R13	The system shall inform if a high score is reached.	Functional	3		Must
R14	The system should have a main menu, selection menu and setting menu.	Functional	3		Could

Use-Cases:

Use Case 1 - Load up the game

- TUCBW the user launches up the executable
- TUCEW the program loads in all assets

Use Case 2 - Generate Main Menu

- TUCBW the user leaves play or first launches the game
- TUCEW the user opens the settings menu, quits the game, or loads up the selection screen

Use Case 3 - Open Settings Menu

- TUCBW the user opens the settings menu from the main menu or pauses.
- TUCEW the user closes the menu, returning to the menu or play screen

Use Case 3.1 - Load Saved Settings

- TUCBW the settings menu is loaded, putting everything to its previous value
- TUCEW everything is properly loaded

Use Case 3.2 - Apply Changed Settings

- TUCBW a setting is changed
- TUCEW a setting is then applied to begin working immediately.

Use Case 4 - Go To Selection Menu

- TUCBW the user opts to play the game.
- TUCEW the user confirms their selection

Use Case 4.1 - Register Selections

- TUCBW the user confirms that the current play conditions are satisfactory.
- TUCEW the play screen is reconfigured alongside the selections and is then loaded in

Use Case 4.2 - Load in additional assets

- TUCBW the play screen needs additional assets
- TUCEW the additional assets have been loaded in

Use Case 5 - Play Screen 1.

- TUCBW user returns from the settings menu or enters from the selection screen. In either case, the AI will begin functionality
- TUCEW users lose the game or pauses, taking them to the main menu or selection screen respectively.

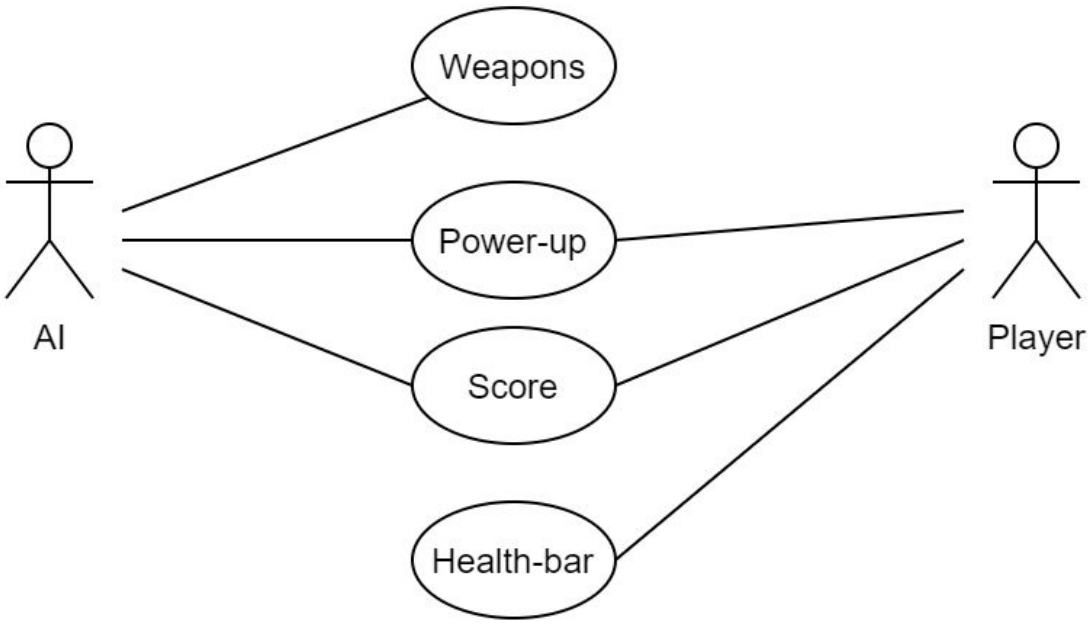
Use Case 6 - Keyboard Input

- TUCBW user presses an interactive key for the game.
- TUCEW the game processes the input and takes the corresponding action

Use Case 7 - Enemy Locomotion

- TUCBW the enemy is created ingame
- TUCEW the enemy collides with the player or with the game border.

Use Case Diagram:



Code Snippet: Full source code in the link below

49 lines (47 sloc) | 1.4 KB

Raw | Blame | History

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Xml;

namespace Carronade {
	public partial class XMLAssetBuilder {
		private ContentManager contentManager;
		private XmlDocument doc;
		public XMLAssetBuilder(ContentManager manager) {
			contentManager = manager;
			doc = new XmlDocument();
		}
		//Since MonoGame doesn't allow explicitly loading all the assets, we have to manually define and track all of our loaded as
		public void LoadAssets(string reference) {
			try {
				doc.Load("Content/" + reference + ".xml");
			} catch (System.IO.FileNotFoundException) {
				Console.WriteLine("oh no. it didn't load. oh well");
				return;
			}
			XmlNode definition = doc.FirstChild;
			if (definition.Name.Equals("asset") && definition.HasChildNodes) {
				foreach(XmlNode asset in definition.ChildNodes) {
					Console.WriteLine(asset.Name);
					Asset newAsset = null;
					switch(asset.Name) {
						case "Image":
							newAsset = BuildImage(asset);
							break;
						case "Sound":
							break;
```

Sample XML

```
<asset>
	<Image>
		<name>loadingText</name>
		<id>0</id>
	</Image>
	<Image>
		<name>loadingAnimation</name>
		<id>1</id>
	</Image>
	<AnimationSet>
		<name>animtest</name>
		<id>2</id>
		<width>16</width>
		<height>16</height>
		<default>rainbow</default>
		<Animations>
			<Animation>
				<name>red</name>
				<start>0</start>
				<end>3</end>
			</Animation>
			<Animation>
				<name>orange</name>
				<start>4</start>
				<end>7</end>
			</Animation>
			<Animation>
				<name>green</name>
				<start>8</start>
				<end>11</end>
			</Animation>
			<Animation>
				<name>blue</name>
				<start>12</start>
				<end>15</end>
			</Animation>
			<Animation>
				<name>blue</name>
				<start>12</start>
				<end>15</end>
			</Animation>
			<Animation>
				<name>purple</name>
				<start>16</start>
				<end>19</end>
			</Animation>
			<Animation>
```

Test Cases

#	Test Case	Test Data	Expected Result	Actual Result	Pass /Fail
1	The Player Avatar responds to input	WASD keys used to test cardinal directions.	W goes up, A left, S down, D right.	W goes up, A left, S down, D right. Combinations nullify each other or add diagonality.	Pass
2	The XML loader reads in XML and produces usable in game assets.	See: canon.xml	Files should be able to refer to ID:100, all animations present	ID:100 exists, works as intended	Pass
3	The Asset Manager should not allow for multiple IDs to collide	.xml asset image made with ID:0 with pre-existing ID:0	Console spits an error and disallows this from happening	Console prints an error, ID:0 stays as is, newly loaded asset is disregarded	Pass
4	The Enemy should track the Player and fire at them	Player Avatar made to run around the canon	Canon's muzzle should follow the player, fire at them	Canon's muzzle follows the player, fires	Pass
5	The R key should completely reset the game and swap the player	R key is pressed	Canon and player both reset to their reset state, player changed for different type	Game resets, player is swapped to a different type	Pass
6	The Blue Enemy should pursue the player	Player Avatar made to run around the map with Blue Enem. onscreen	Blue Enemies should slowly turn towards the player and follow.	Blue Enemies Chase the Player	Pass

7	The Invincibility Powerup should render the player impervious to damage	Player Avatar picks up the invincibility powerup	Enemies that collide with the player should have no effect	Player takes no damage from the exchange	Pass

Customers and User:

All the users who will play this game are our customers. For testing and getting feedback we provided our colleagues with the game and asked for constant feedback from them about the features we implemented in each of the iterations. To gather feedback we have provided an online form which asks several free response questions to the user concerning the gameplay, flow of the program, functionality check, errors, and bugs.The users who provide the feedback have some knowledge on different types of games and their gameplay. In case of faults and bugs in the software we set up a meeting with users virtually to get in depth knowledge about the errors and fix them along the way.

Github Link:

https://github.com/udhungana/team7cse3311

Carronade Version 0.2