Table of Contents

[2 Analysis 3](#_Toc510188369)

[2.1 Identification And Background 3](#_Toc510188370)

[2.2 Identification Of User Needs And Acceptable Limitations 3](#_Toc510188371)

[2.3 Overview Of Current Systems 5](#_Toc510188372)

[2.3.1 Suitability of Current Systems 6](#_Toc510188373)

[2.3.2 Detailed Analysis of Selected Current Systems 6](#_Toc510188374)

[2.3.3 Summary of Features from Researched Games to be included 9](#_Toc510188375)

[2.4 Objectives For The Proposed System 9](#_Toc510188376)

[2.5 Hardware and Software Limitations 10](#_Toc510188377)

[2.6 Other Limitations 11](#_Toc510188378)

[2.7 Justification Of Chosen Solution 11](#_Toc510188379)

[3 Design 12](#_Toc510188380)

[3.1 Overview of Entities 12](#_Toc510188381)

[3.2 Entity Relationship Model, Diagrams And Descriptions 13](#_Toc510188382)

[3.3 System Flow Chart 14](#_Toc510188383)

[3.3.1 Detail system flowchart – Explosion “Causes damage to other tanks” 15](#_Toc510188384)

[3.3.2 Detail system flowchart – AI Tank “Hunts other tanks/moves evasively” 15](#_Toc510188385)

[3.3.3 Detail system flowchart – Game Manager “decides game end reached” 16](#_Toc510188386)

[3.4 Object Analysis Diagrams – Inheritance And Aggregation 16](#_Toc510188387)

[3.5 Data Volumes 16](#_Toc510188388)

[3.6 Data Sources 16](#_Toc510188389)

[3.6.1 Game Menu Settings (Inputs) 16](#_Toc510188390)

[3.6.2 Player Controls (Input) 16](#_Toc510188391)

[3.6.3 Other Controls – from Game Menu (Input) 17](#_Toc510188392)

[3.6.4 Other Controls – from Game Menu (Output) 17](#_Toc510188393)

[3.7 Data Flow Diagram 17](#_Toc510188394)

[3.8 Human-Computer Interface 17](#_Toc510188395)

[3.8.1 Player game controls: 17](#_Toc510188396)

[3.8.2 Usability feature 18](#_Toc510188397)

[3.8.3 Menu Screen Interface 18](#_Toc510188398)

[3.9 Class Definition Diagrams, Detail Of Object Behaviours And Methods 19](#_Toc510188399)

[3.9.1 Computer Tank Class 19](#_Toc510188400)

[3.9.2 Player Tank Class 20](#_Toc510188401)

[3.10 Data Dictionary 20](#_Toc510188402)

[3.11 Identification Of Storage Media 21](#_Toc510188403)

[3.12 Description Of Record Structure 22](#_Toc510188404)

[3.13 Validation Required 22](#_Toc510188405)

[3.14 Test Strategy for Iterative Development 22](#_Toc510188406)

[3.15 Test Data for Iterative Development 22](#_Toc510188407)

[3.16 Test cases for Iterative Development 23](#_Toc510188408)

[3.17 Post development testing 25](#_Toc510188409)

[4 Development and Testing 26](#_Toc510188410)

[4.1 Iteration #1 - PROTOTYPE 26](#_Toc510188411)

[4.2 Iteration #2 27](#_Toc510188412)

[4.2.1 Objectives 27](#_Toc510188413)

[4.2.2 Test plan 27](#_Toc510188414)

[4.2.3 Game Arena 28](#_Toc510188415)

[4.2.4 Player tank movement 29](#_Toc510188416)

[4.2.5 Player shooting 30](#_Toc510188417)

[4.2.6 Spawning enemy tanks 30](#_Toc510188418)

[4.2.7 Computer tank movement logic and shooting 31](#_Toc510188419)

[4.2.8 Results of Testing 33](#_Toc510188420)

[4.2.9 Stakeholder feedback 34](#_Toc510188421)

[4.3 Iteration #3 34](#_Toc510188422)

[4.3.1 Objectives 34](#_Toc510188423)

[4.3.2 Test Plan 35](#_Toc510188424)

[4.3.3 Different Tank Sizes 35](#_Toc510188425)

[4.3.4 Area effect damage for shells 36](#_Toc510188426)

[4.3.5 Fix the memory leak 37](#_Toc510188427)

[4.3.6 Results of Testing 38](#_Toc510188428)

[4.3.7 Stakeholder feedback 38](#_Toc510188429)

[4.4 Iteration #4 38](#_Toc510188430)

[4.4.1 Objectives 38](#_Toc510188431)

[4.4.2 Test Plan 39](#_Toc510188432)

[4.4.3 Game Menu 40](#_Toc510188433)

[4.4.4 Split screens for multiplayer games 41](#_Toc510188434)

[4.4.5 Results of Testing 41](#_Toc510188435)

[4.4.6 Stakeholder feedback 43](#_Toc510188436)

[5 Evaluation 43](#_Toc510188437)

[5.1 Requirement Check 43](#_Toc510188438)

[5.2 Feedback from stakeholder - Jeremy Erwen 44](#_Toc510188439)

[5.3 Level of Success of usability features 44](#_Toc510188440)

[5.4 Issues, Maintenance Issues and Limitations of solution 44](#_Toc510188441)

[5.5 How partially met/unmet criteria and limitations could be addressed by further development 45](#_Toc510188442)

[6 ALL TANK TERRITORY CODE 45](#_Toc510188443)

# Analysis

## Identification And Background

Third person shoot-em-up games are incredibly popular. Among these games, Tank simulation games have mostly followed a different path. A tank simulation game follows different movement rules to other 3d shoot em ups – there are four basic movements (forward, backward, rotate left and rotate right) that can be combined into a total of eight movement – but movement such as ‘move sideways’ and ‘move diagonally’ cannot be performed. This is because at their heart tank simulation games operate on the principle that in real life there are two separate motorised tracks that govern movement, and their only abilities are to (independently) move forward, backward, or remain static.

Tank games have evolved along a super-realistic path, in many cases including Massively Multiplayer games of up to 16v16 tanks. In almost all cases only one player can use the PC while playing .What does not appear in any of the published games is a lightweight (in terms of memory footprint and storage space), fast-moving game that can be enjoyed by multiple players simultaneously. This is, presumably, because in the race for increased realism, there has been an increase in the number of controls required – so in the super-realistic simulations tank commanders can repair damage to their tanks, fire chaff, lay down smokescreens, and much much more. As a result more keystrokes have to be reserved to specific actions, meaning there aren’t enough remaining to make them a single-PC multi-player game.

The stakeholders have requested a tank game that can be played by up to 4 players on the same PC. It must be fast-moving (meaning frantic action as well as a short time limit). The game should appeal to children more than military historians, so cartoonish graphics are to be preferred. Since 4 players may be crowded around the same keyboard, the keystroke combinations available to them should be enough to simulate tank movement, as above, and allow the tanks to fire shells, but nothing more.

## Identification Of User Needs And Acceptable Limitations

My main stakeholder is Jeremy Erwen, a 47 year old male. He used to be a gamer, but has limited time available to play computer games any more. My second stakeholder is Idris Erwen, a 14 year old male with autism. He has a limited attention span. The main stakeholder has given me requirements on his and Idris’s behalf.

This is a paraphrased conversation I had at the start of the project with the main stakeholder (MS). I have tagged any strong preferences from the stakeholder with a tag that it is an objective, and these are presented later in the document:

(Me): Why do you like tank games?

(MS): I like the tank movement, the physics of it. Because you can’t move sideways, jump, or go in diagonals, your tank is a lot easier to hit. I’ve played multiplayer infantry shooters where half the players are jumping diagonally across the field while still shooting at you. It’s not realistic, and as I get older and my reactions get worse, it’s very frustrating.

(Me): What tank games have you already played, and what did you like and not like about them?

(Main Stakeholder): I’ve played World of Tanks recently, and a long time ago I used to play Battlezone in the arcades.

World of Tanks looks very realistic but is very complicated – plus it is a pay-to-upgrade model. It takes a long time to get good tanks, and it is frustrating to think you’ve done well, only to be blown to bits by someone who has payed for upgrades. Also the games go on too long for me.

(Me): So you’d like a faster, shorter and simpler type of game? [OBJ2][OBJ3]

(MS): Exactly. And something a lot more frantic – maybe loads of tanks occupying a small space. In World of Tanks you can spend 5 minutes moving up to the front, and get shot to death from a long distance by an enemy you can’t even see.

(Me): What about Battlezone?

(MS): Battlezone felt very advanced at the time, but there were parts to it I didn’t like. It was a first person shoot em up, so if an enemy tank was near you and you drove past it, you’d have no idea where it went to. World of Tanks’ 3rd person view is much better. [OBJ4]

(Me): World of Tanks is a Massively Multiplayer game. Do you want this to be multiplayer?

(MS): Yes, but I’d like computer tanks as well. But I don’t want to play a multiplayer game where I’m in one room and Idris is in another. [OBJ6]

I played games before where two players used the same keyboard – Gauntlet, Spy vs Spy, and a couple of racing games. Spy vs Spy and the racing games had split screens. [OBJ6] This was great fun, because everyone was crowding round the same keyboard, and it made for a very fun atmosphere. Could we all use the same keyboard? How many people could we fit around one keyboard? [OBJ6]

(Me): I don’t know, I’ll have to find out!

World of Tanks has very detailed graphics, but Battlezone on the other hand has very basic wireframe graphics. How detailed are you expecting the game to be?

(MS): I’m thinking this should be a fast, frantic game [OBJ2], so probably more cartoonish graphics would be better. I definitely don’t need the level of detail you get in World of Tanks.

(Me): Are there any other requirements you can think of?

(MS): Can we have different types of game? Maybe one where you keep going until there’s only one tank left, and another with a short time limit where you try to get the best score? [OBJ7][OBJ11]

If there’s a game with scoring, I’d like to have a high score table too. [OBJ11][OBJ12]

Idris will play this game with his cousins when they visit, to encourage him to be more sociable. The other children range in ages from 8 to 16, so there should be a way of handicapping some players so that they all have a decent chance to win. [OBJ15]

## Overview Of Current Systems

A review of online resources showed a large number of tank games have been commercially produced as PC games or video games. I will not discuss many of the games because they were all 1st person games with no multiplayer elements. These were

Abrams Battle Tank (1991), Arcticfox (1986 ), Assault (1988 arcade video game), Battle Tank (1990 ), Panther (1975 video game), Recoil (1999), Stellar 7 (1983), Tank (1985 arcade game), Tank Force (1991 arcade game), Tokyo Wars (1996 arcade game).

I have also excluded from this analysis tank games that were really about team management, such as iMIA1 Abrams (1997).

Tank games which had a 3rd person perspective or multiplayer element are:

Armored Fist 3

This was a 1999 first person tank game, with a multiplayer option via the internet. It was not well received (single player version was described as outdated, and the multiplayer element was felt to be too boring). Players had the whole screen to themselves in multiplayer mode.

Armored Warfare

This was a 2015 tank game by Obsidian Entertainment. Graphically, it is a highly realistic tank game, with 1st or 3rd person views, and designed to be both multiplayer and have single player ‘missions’. Several tanks are available initially and others are unlocked on progressing through the game or paying for credits (it is a free game with a ‘pay to upgrade’ revenue model). There is one player to a screen.

BattleTanx

This was a 1998 tank game by the 3DO Company. It was quite advanced for its time, featuring a choice of tank, and multiplayer games over the internet. It also had four different game modes, but was 1st person perspective and there were no shared screens.

Battlezone (1980 arcade video game and 1998 PC video game)

This game will be reviewed further in the Detailed Analysis section.

Iron Warriors: T-72 Tank Commander

This was a 2004 game by Crazy House. It had realistic physics, and the ability to switch between various tank positions (gunner, driver, machine gunner and commander). It had a 1st or 3rd person view and had an interesting feature where in multiplayer mode players could occupy different positions in the same tank. Each multiplayer, however, played on a full screen.

Mechwarrior Online (2014)

Not exactly a tank game – the vehicles used are robot bipeds so they don’t follow tank movement rules – but a very interesting game . It has 1st and 3rd person views, multiplayer action via internet, and split screen multiplayer game option so players can use the same keyboard.. It also has multiple game modes.

Panzer Commander

A very advanced game for 1998, Panzer Commander allowed the player to choose any one of 10 different tanks, different game modes and multiplayer functionality. Multiplayer games were one full screen per player.

Tank Troopers

This is a 2016 Nintendo 3DS 3rd person tank game, with a choice of up to 30 tanks and internet multiplayer games for up to 6 players. There was also a single player mode.

World of Tanks

This game will be reviewed further in the Detailed Analysis section.

### Suitability of Current Systems

From the interview with the main stakeholder, there are five gameplay requirements that the stakeholder wants. – Tank movement, multiplayer game, split screen, 3rd person perspective, and different game modes. A summary of the systems reviewed shows that there is no suitable commercial product which meets these requirements.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Product Name** | **Tracked Tank Movement** | **Multiplayer available** | **Split Screen** | **3rd Person View** | **Multiple Game Modes** |
| Armored Fist 3 | Yes | Yes | No | No | No |
| Armored Warfare | Yes | Yes | No | Yes | Yes |
| BattleTanx | Yes | Yes | No | No | Yes |
| Battlezone | Yes | No | No | No | No |
| Iron Warriors | Yes | Yes (special) | No | No | Yes |
| Mechwarrior Online | No | Yes | Yes | Yes | Yes |
| Panzer Commander | Yes | Yes | No | Yes | Yes |
| Tank Troopers | Yes | Yes | No | Yes | Yes |
| World of Tanks | Yes | Yes | No | Yes | Yes |

### Detailed Analysis of Selected Current Systems

Two systems have been selected for deeper analysis – Battlezone - an arcade game from 1980 – because it was the first use of a two-track tank movement system, and World of Tanks (2012) because it is modern and has all the required stakeholder features apart from split screens.

Battlezone (Atari) - 1980



Battlezone was a single-player arcade tank game, made by Atari Inc.



The game used wireframe vector graphics, which are abstract. The game used first person perspective to make up for this, as it is easier to see what is going on because the player only sees one tank on screen at a time.

Having only one tank be onscreen at a time made the game easier, as there was only one tank firing at the player; hence, the game does not feel unfair as the incoming shots are fired at the same velocity and frequency as the player’s shots. Shooting a tank awards 1000 points.

Additionally, every tank can only shoot one bullet at a time. This makes it easier for the player to dodge incoming bullets from the enemy tank; at the same time, it prevents the player from beating the game easily by rapidly firing bullets, and encourages the player to aim carefully at the enemy tank. *This is an interesting feature which I will incorporate into the game, rather than having a fixed ‘reload time’ after firing. It will encourage careful aim, and will also reduce the memory usage of the game since multiple shells flying around per tank could increase memory usage significantly.* *[OBJ15 later]*

Instead of the player having health points or a health bar, it has three lives. If the player is hit by a bullet, they lose a life. The game ends when all lives has been lost, and if the player has beaten the previous high score, the new high score is stored in the game’s database and is displayed below the player’s score in subsequent games. This encourages the player to play again to try and beat their score. *The stakeholder has already requested a high-score feature for the new system.*

The controls mirror the two tracks of a tank – one joystick controlled the forward-back movement of the left track, and the other joystick controlled the right track. This means the tank can move straight forward and back, rotate on the spot, or move forward/back in a curve.

There is a radar above a battle screen, in the shape of a clock. The radar shows the location of the player and any enemies in range, but it is a bit too abstract. *This will be unnecessary with a 3rd person view which the stakeholder has requested.*

On the top left corner of the screen, there is text that tells the player if an enemy is in range, and whether to turn left or right to face the enemy. This is helpful for the player to understand what they need to do, and makes the game more accessible.

There are indestructible geometric shapes on the screen, that can act as either obstacles to the player, or as structures the player can hide behind to defend themselves. *Having obstacles on the game arena seems like a good idea, as it gives cover and affects freedom of movement. This would make the game more tactical than a big shootout in an empty arena. [OBJ2]*

**World of Tanks (2012)**



World of Tanks has realistic models, and exciting multiplayer gameplay. It operates on a ‘pay to upgrade’ model, which provides its revenue stream and means there is a very large variety of tanks available.

It is possible to switch between first person and third person views.



Tanks have a maximum health, and actions like being hit by shells, running into other tanks, or running into obstacles at high speed result in damage represented by loss of health. If the tank’s health reaches zero, it dies, and respawns at the start area. *This is a feature which should be incorporated for good gameplay. It is not fun to die after being hit once.*

The game runs to a 15 minute maximum time limit, and has victory conditions (such as holding enemy areas for prolonged times) that can end the game early. *While the stakeholder has asked for a time limited game, I don’t think he was asking for as long a time limit as 15 minutes. He wanted quick, fast games.*

It is a Massively Multiplayer Online (MMO) game – there are multiple player tanks per game, and the game can be played on PC and mobile.

There are a variety of weapons available - reload time is quite long, depending on the weapon equipped. There is a rough correlation between how much damage a weapon will do and how long it takes to reload.

The tanks are classified as Heavy, Medium and Light. Heavy tanks move slowly but have high health and powerful weapons, whereas light tanks are very manouverable but lightly armoured. They are unlikely to survive going toe to toe with a Heavy tank, but can move quickly enough to be difficult to hit from a distance. *Having different classes of tank is a good idea for variety, and I will attempt to include it in the game, at least for the AI tanks.*

Controls Arrow keys are used on the PC, and on the mobile version an 8-point arrow pad is provided. Both provide the same manouverability (also the same as for Battlezone) - forward, backward, rotate on the spot, and curve forwards and backwards.

### Summary of Features from Researched Games to be included

From the above researched games desirable features, we want to make a game with tank movement, and multiplayer. The 3rd person perspective view as in World of Tanks seems more appealing that 1st person in Battlezone. None of the tank games researched had a split screen feature, which will be challenging but will be a unique feature of the game.

## Objectives For The Proposed System

These are the high-level objectives for the system, against which the finished product will be evaluated.

These high-level objectives have been taken from: -

* the stakeholder requirements
* common features of all good tank games
* best features, where applicable, of the other tank simulations analysed.

|  |  |
| --- | --- |
| **Ref** | **High Level Objective and *Justification*** |
| OBJ1 | The game must be playable on a Windows PC. |
| OBJ2 | The game area should be small, with obstacles.  *This is because the stakeholder has asked for a fast-paced, lively game. A small game area with obstacles should make it possible to avoid fire, but not possible to hide.* |
| OBJ3 | There should be a time limit on the games  *This has been requested by the stakeholder. The full request is for a short time limit, but as a high level objective time limits on games is sufficient..* |
| OBJ4 | The perspective should be 3rd person (“above and behind”) for the players.  *Required by the stakeholder.*  *With a small game area, the player tanks will be fired upon from all angles. 3rd person perspective gives the player a small chance of spotting and avoiding a shot from behind.* |
| OBJ5 | Tank movement should be as per two-track vehicle movement – i.e. tanks should be able to move forward and backwards, rotate in either direction, and move in an arc. They should not be able to move diagonally or sideways.  *This is a fundamental part of tank games.* |
| OBJ6 | There should be the option for 1 to 4 players in the same game using split screens, and all player actions coming from the same keyboard.  *As shown in the analysis of other systems, this is the unique selling point of the game. All the players have to gather around the same keyboard.* |
| OBJ7 | There should be two game modes. These are Survival, where the players accumulate scores for doing damage to the other tanks and try to last to the time limit, and Last Tank Standing, where the objective is to be the sole remaining tank.  *This introduces two distinct playing styles. In Last Tank Standing, the players should be defensive and concentrate on avoiding taking damage. In Survival, the best approach would be to be aggressive and get a big score by damaging lots of tanks.* |
| OBJ8 | Computer-controlled tanks should operate on line of sight, and be hard to hit.  *The game should be challenging as well as fun. AI tank movement logic should be evasive, and the tanks should not get ‘stuck’ against obstacles as this would make them easy to hit.* |
| OBJ9 | All tanks can fire shells  *This is a common feature of all tank games.* |
| OBJ10 | Shells explode on collision with obstacles and other tanks, causing damage to every tank within the explosion radius.  *The early tank games either required a direct hit, or did not actually create a shell that travelled down its firing path. This is not realistic. A shell hitting a wall right next to a tank should still cause damage to the tank.*  *This means it is possible for a player to blow themselves up, if they fire at enemies from a point-blank range – their own tank will be caught up in the blast radius.* |
| OBJ11 | For Survival game mode, there should be a scoring system for doing damage to other tanks.  *This won’t be necessary for Last Tank Standing, because the order in which tanks are destroyed would determine the rankings.* |
| OBJ12 | There should be a high score table for the best Survival scores.  *As requested by the stakeholder, who would like to record what the best scores were.* |
| OBJ13 | Tanks should be able to survive more than one direct hit.  *It is more enjoyable if the player and enemy tanks are fairly durable, and this also means I could put a healthbar on each player’s screen, so they know who is damaged and so vulnerable to attack.* |

I will add one objective to the list. Considering the wide range of ages and ability of the users of this system,

|  |  |
| --- | --- |
| OBJ14 | There should be a Difficulty setting for Player Tanks  *This is so that all players can have a chance to win. An 8 year old or a person with autism will find it very difficult to win a game where the other players are teenagers or adults, and we want to make the system fair and fun for everyone.* |

## Hardware and Software Limitations

The stakeholder requirement to have as many players around the same keyboard as possible. From practical inspection this appears to be 4 players – one player to the left using WASD keys for movement and Left Shift to fire, one in the middle using GYHB and the Spacebar, one to the right using the arrow keys and the right Ctrl key, and to the far right one player using the number pad.

Not all keyboards have a separate number keypad, so on laptops the maximum number of players will be three, and possibly only two if the arrow keys are also in a difficult position. [LIM1]

The game will have a fixed number of tanks, which may decrease over time depending on the game mode. The amount of memory the game uses could increase dramatically, however, if I do not limit the number of shells that can be fired - since each shell has a memory usage of its own, plus a further memory usage when the shell collides with something and I create the explosion effect with it. [LIM2]

## Other Limitations

Because I have only four months to develop the game, I may not be able to implement every feature I desire in the game. I may not be able, for example, to include multiple game modes. I will work towards making a fully working game, and implement additional features only if I have time left over including time for testing.

## Justification Of Chosen Solution

I will develop the game in Unity, using C# for the scripting. Unity is a game development platform that handles a lot of game physics and collisions for you; in return you have to learn about Vectors (Movement in 3d space) and Quaternions (Rotations across 3 axes).

Unity code runs asynchronously – any object with an Update() method gets called 30 times a second. This means I can have a keyboard movement system which can pick up multiple simultaneous keystrokes. So I will be able to pick up on a user keystroke to move forward as well as one to rotate at the same time, giving the right type of movement as a result, and do this for up to 4 players.

Unity is object oriented, so with the right design I will be able to write generic code for the Computer tanks, and once instantiated they can all operate independently – hunting, shooting and evading - of any central controller.

The Unity platform has a Unity Store with lots of pre-fabricated game objects, meaning I can pick a tank object to use instead of spending lots of time making them myself.

Finally, Unity is a universal code base meaning I could port the code onto any platform without having to rewrite anything – so I could recompile the code to make the game playable on a Mac if I wanted to.

# Design

The task laid out in the Analysis phase is for a multiplayer tank game with split screens, and all players using the same keyboard. Since one of the stakeholders has special needs and may be playing alongside other people ranging from young children to the 47 year old main stakeholder, I think it would be a good design decision to have a difficulty setting that can be applied to each player’s tank. This would increase or decrease the tank’s health, speed, shot power and shot speed, depending on a setting of easy, medium or hard.

One of the strengths of Unity is that it supports asynchronous operations. Any script which has an Update() method in it will have that method called every 30th of a second. The effect of this is that game objects, once created, can operate independently of the instructions of any central controlling object.

I will take full advantage of this, and so the hierarchy and system flow will be more event-driven than sequential. In this Design phase I will outline all the major components of the game, and describe how they will interact.

## Overview of Entities

The main entities required for the solution are a Menu Screen, Game Manager, the Arena, Player Tanks, AI Tanks, Shells, and Explosions.

**Menu Screen**: An interactable dialog box, which allows the user to set the game type to be played, its duration, the number of players and the number of AI tanks. These settings are fed back to the Game Manager and used in setting up a new game. The Menu Screen also provides the means to view High Scores, and Exit the game.

**Game Manager**: This object is a set of scripts, and is the core of the system. It starts the game in line with the settings provided on the Menu Screen. It instantiates the player and AI tanks, keeps track of the time, and keeps a record of the score.

**Arena:** This is thearea the game takes place in. By design it will be quite small, so that the game will feel very busy and crowded with all the player and computer enemy tanks. There will be obstacles in the arena, and all the arena walls and obstacles will be barriers that tanks cannot move through, and shells explode with on collision.

**Player Tanks**: Once instantiated by the Game Manager, Player tanks respond to user (keyboard) input, and are destroyed once they take too much damage. They can fire shells, which results in the creation of a Shell object. Player Tanks have their own Health value.

**AI Tanks**: Also instantiated by the Game Manager, AI tanks move according to their own logic. They hunt for other tanks to attack when they are able to fire, and move evasively if they are unable to fire. They are destroyed once they take too much damage. AI tanks also fire shells, which also results in the creation of a Shell object.

**Shells**: Once fired by a Player Tank or AI Tank, shells are independent of who fired them. The only link is that a Shell’s name starts with the name of the tank that fired it. This will let me work out which tanks get points for damaging other tanks. Shells are not child objects of Tanks. If they were, then destroying a tank object (in scripting terms) would also cause the Shell object to disappear mid-flight.

**Explosions**: when Shells collide with Tanks or other obstacles in the game arena, they explode making a visual effect and causing damage to all tanks in the blast radius. This damage is then reported back to the Game Manager for score keeping.

## Entity Relationship Model, Diagrams And Descriptions

In visual terms the relationship between the entities can be outlined as follows

**Game Arena**

Constrains Tanks

Records Spawn Points

**Game Manager**

Loads user Settings

Runs Timer

Keeps Score

Initialises Game

Updates High Scores

**Settings Menu**

Set game settings

View High Scores

**AI Tanks**

Hunt other tanks

Evade other tanks

Fires shells

**Player Tanks**

Player driven

Fires shells

**Shells**

Explode on contact with other objects

**Explosions**

Cause area effect damage

Shows explosion effect

## System Flow Chart

The following is a high-level flow chart for the system. Because the Game Manager is accessed in several places in the system flow, it appears more than once in the diagram. The activities and events are explained in greater detail later in this document.

Clicks Show High Scores

Causes damage to other tanks

Hunts other tanks

Moves Evasively

Player Tanks

Create Game objects

Create Game objects

Repeat

Start

Game Manager

Menu Screen

High Scores

AI Tanks

Shell

Shell

Explosion

Game Manager

Explosion

Game Manager

Menu Screen

Displays

Sends game settings

Fires at enemy

Collides with tank or wall

Causes damage to other tanks

Fires at enemy

Collides with tank or wall

Start Game

User Controlled movement

Game end reached

Sends game results

Local storage

Load

Update scores

Update scores

### Detail system flowchart – Explosion “Causes damage to other tanks”

It is important to know who has caused damage with a shell, since we score a Survival game by who has dealt the most damage. This event in fact does two things – damages an affected tank, and credits a score to the player who dealt the damage.

Create a radius around the shell impact

Does the radius intersect any tanks?

Calculate distance: shell to tank

Damage = shell power / distance

Apply damage to struck tank’s health

Determine shell owner

Credit Damage to shell owner’s score

Start

No

Yes

End

### Detail system flowchart – AI Tank “Hunts other tanks/moves evasively”

Computer tanks have two behaviours – hunting and evasive manouevering. This is because the game design will allow only one live shell per tank (see discussion in Analysis, of the game Battlezone). As a result, the computer tank will seek out enemies when it can fire, and concentrate on avoiding getting hit when it cannot fire.

Yes

Yes

No

Yes

Can I fire?

Can I see a tank?

Is it close?

Fire

Close in on tank

Circle

Move evasively

Start

No

No

“Circle” means to move in a big circle (a combination of move forward and rotate). “Close in” is to move forward in a straight line. “Move evasively” should be a random combination of moving forward or backward, occasionally reversing direction and/or turning.

### Detail system flowchart – Game Manager “decides game end reached”

The game should not end only when the timer runs out. It should also come to an end when there are no user players left in the game.

No

Are any players left?

No

Start

Has timer ended?

Yes

Yes

End game

## Object Analysis Diagrams – Inheritance And Aggregation

## Data Volumes

Stored data volumes should be minimal. High scores will be recorded to the PC hard drive at the end of a game of Survival, and will be accessible to be viewed when the user chooses to view previous high scores from the game menu.

## Data Sources

Input data source will be keyboard input for all player tank movement and firing. Game menu options will be accessible by mouse click. For High Scores, there will be output to and input from a locally stored file.

### Game Menu Settings (Inputs)

* Set number of players
* Set number of AI tanks
* Set duration of game
* Set game mode – Last Tank Standing or Survival

### Player Controls (Input)

For each tank – keyboard input controlling

* Forward
* Reverse
* Rotate Left
* Rotate Right
* Fire shells

### Other Controls – from Game Menu (Input)

* Open Game Menu – Keyboard input (Escape key)
* View High Scores
* Start/restart game
* Continue Game

### Other Controls – from Game Menu (Output)

* Record High Score

## Data Flow Diagram

Display High Scores

Start

Game Menu Options

User Settings

Mouse

Read Local File

View High Scores

Player Actions

Keyboard

Yes

Continue until game end

Populate Tanks

Run Game

Start Game

New High Score?

Write to Local File

Display High Scores

## Human-Computer Interface

There will be two types of human-computer interface: player game controls, and the use of the Menu Screen to change game settings and view high scores. An appropriate set of key bindings for 4 players would be:

### Player game controls:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Player number | Forward | Reverse | Turn Left | Turn Right | Fire |
| 1 | W | S | A | D | Left Shift |
| 2 | Y | H | G | J | Spacebar |
| 3 | Arrow Up | Arrow Down | Arrow Left | Arrow Right | Right Ctrl |
| 4 | Numpad 8 | Numpad 5 | Numpad 4 | Numpad 6 | Numpad 0 |

### Usability feature

There may be a hardware limitation on using a laptop keyboard (see Analysis, [LIM1]). The key bindings for player game controls have been chosen so that a 2 player game will use a set of keystrokes that are always available on any keyboard.

### Menu Screen Interface

This is a draft of the interface to change all the game settings.



This should ideally be driven with the mouse. The options are all dropdowns or button clicks, so mouse interaction makes a sensible choice.

Through this interface the game mode, time limit, number of players and number of computer players can be set.

## Class Definition Diagrams, Detail Of Object Behaviours And Methods

Class definition diagrams are provided for Computer Tanks, with their associated Shells and Explosions, and for Player Tanks

### Computer Tank Class

**AI Tank**

Shot Power

Shot Speed

HasLiveBullet

BeAggressive()

MoveEvasively()

Shoot()

**BigGreyTank**

BigGreyTank prefab

Health

Speed

**Shell**

Shell prefab

ShellName

Shot Power (inherited)

Speed (inherited)

OnTriggerEnter()

**Explosion**

Explosion Prefab

Play()

**GreyTank**

GreyTank prefab

Health

Speed

**LittleGreyTank**

LittleGreyTank prefab

Health

Speed

Instantiates on Shoot()

Shells are not children of the AI Tank object, but because they instantiate them on firing I have included them in the diagram. They are not child objects of tanks so that a shell fired by a tank that is then destroyed will not result in the shell also being destroyed.

ShellName, a variable of Shell, will retain the name of the tank that fired it this is important when calculating damage. If we can work out the name of the firer after an explosion and it was a player tank, then we can call that tank’s public method (in this case DoDamage(), below) to credit its score.

HasLiveBullet is important – because the design allows for one live bullet only, the Boolean HasLiveBullet determines whether a tank can fire, which in turn decides whether it will be aggressive or defensive.

### Player Tank Class

**PlayerTank**

Health

Speed

Shot Power

Shot Speed

HasLiveBullet

Score

TankMovement() TakeDamage()

Move () DoDamage()

Turn() Shoot()

Shoot()

**Shell**

ShellName

Shell prefab

Shot Power (inherited)

Speed (inherited)

OnTriggerEnter()

**Explosion**

Explosion Prefab

Play()

**Tank Prefab**

Player Tank prefab

Instantiates on Shoot()

TankMovement() is an input function that scans the keyboard for pressed keys, and translates those keystrokes into Move(), Turn(), and Shoot() actions.

TakeDamage() and DoDamage() are public functions related to exploding shells. TakeDamage() will reduce the tank’s health. It is called by a shell that explodes and does damage to everything around it. DoDamage() is a method of crediting a player’s Score with whatever damage their exploding shell has caused.

## Data Dictionary

**Key variables and structures**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field name** | **Variable Owner** | **Description** | **Data type** | **Length** | **Sample** | **Validation** |
| **Health** | Player tanks / AI tanks | Stores how many health points the player has. | Float | 4 digits | 29.53 | Must be greater than 0 for live tanks |
| **Speed** | Player tanks / AI tanks | Stores the speed of tanks | Float | 2 digits | 20.0 | Must be a real number |
| **HasLiveBullet** | Player tanks / AI tanks | Records whether the tank has already fired a bullet | Boolean |  | False |  |
| **TurningSpeed** | Player tanks/AI tanks | Stores the rotation speed of tanks. | Float | 3 digits | 105.01 | Must be greater than 0 |
| **myTurningDirection** | AI tanks | Stores the current direction of tank rotation (clockwise/ anticlockwise) | Float | 1 digit | 1.00 |  |
| **RedTankScore, BlueTankScore, GreenTankScore, YellowTankScore** | Game Manager | Stores the players’ scores. | Integer | 3 digits | 35 | Reset to 0 at start of game. |
| **ShellName** | Shell | Stores the name of the tank that fired the shell. | String | N/A | “RedBullet” “GreyTank1 Bullet” | Must start with a player colour, or with the name “GreyTank”, so that stripping off “Bullet” at the end reveals the tank object that fired. |
| **ShotPower** | Tank, and passed on to Shell public variable | Stores the amount of damage that a shell can do. | Float | 4 digits | 15.46 | Must be greater than 0. |
| **ShotSpeed** | Tank, and passed on to Shell public variable | Stores the speed at which a shell travels | Float | 4 digits | 15.46 | Must be greater than 0. |

## Identification Of Storage Media

A small file will need to be stored in the same location as the game’s executable. This is to record the high scores. It is suggested storing in the same location as the executable is best because it makes it less likely that reading or writing to the file will encounter security restrictions.

## Description Of Record Structure

A very simple, comma-delimited record structure is proposed:

|  |  |
| --- | --- |
| **Filename:** | **HighScores.txt** |
| **Score** | Numeric |
| **PlayerName** | Text, up to 12 charcters |

The reason for recording the Score before the PlayerName is that a player entering a comma as part of their name will not break the system or require more levels of validation before the data is loaded in. If it was the other way around, a player with a score of 123 and entering “Smith,John” as their name would end up passing “Smith” as a playername, and “John123” as a score.

## Validation Required

The Menu Screen settings and entering High Score data are the areas that will need validation.

|  |  |  |
| --- | --- | --- |
| Object | Activity | Validation |
| High Score Screen | Enter new high score | Maximum of 12 characters. Empty string also acceptable. |
| Menu Screen | Change Time Limit | Value should be restricted to 1-3 minutes |
|  | Set Number of players | Value restricted to 1-4 players |
|  | Set Number of computer tanks | Value should be restricted to 5-20 tanks |
|  | Set Game Mode | Only options should be ‘Survival’ and ‘Last Tank Standing’ |
|  | Set player difficulty | Only dropdown options should be ‘Easy’, ‘Normal’ and ‘Hard’ |

## Test Strategy for Iterative Development

During each development cycle, high level objectives will be addressed in line with the Design. Tests will be devised that (i) test the appropriateness of the solution to the high level objective, and (ii) confirm that the Design criteria have been met

## Test Data for Iterative Development

In addition to meeting the high level objectives, the success of the game will be determined by how enjoyable the gameplay is. While we can’t quantify ‘fun’, I will use stakeholder sessions to get feedback on how the game hangs together as a whole.

Unity allows public variables in scripts to be set at design time in a dialog box. By setting variables that directly affect gameplay, I can run a stakeholder session to play the game in debug mode, and tweak the game variables until there is a good balance in the game. This may be challenging. If, for example, we increase all shell shot power massively, then everything will die in one hit. But then the question becomes do we increase all tanks’ health to compensate, or reduce the shot power or size of the explosion blast radius?

The gameplay variables that will be reviewed will be:

Tank: Health, Speed, turningSpeed

Shell: shot power, shot speed, blast radius.

## Test cases for Iterative Development

These are the test cases that will be used during iterative development. To show justification for the choice of tests reference has been made back to the Objectives from the Design phase.

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Statement | Test | Expected result |
| 1 | Player tanks move forward, back, left and right in response to key presses. **OBJ5** | For each tank, check the movement keypresses in the Design section are correct. | Movement in line with Design document expectations |
| 2 | Player tanks move appropriately when a combination of move keypresses are made. **OBJ5** | For each tank, press combinations of Forward/back, and left/right | Tanks should move in circles |
| 3 | Player tanks fire in response to its shooting keypress **OBJ9** | For each tank, press the Fire key | Tank should shoot |
| 4 | The game arena is a hard obstacle, which stops shells and tanks **OBJ2** | Run a player tank into all the walls | The tank should be blocked by the wall |
| 5 | Every tank can only have one live shell at a time. **OBJ14** | With a player tank, fire and then attempt to fire again | No second shell should be fired |
| 6 | If a computer tank cannot fire, it will move evasively. **OBJ8** | Set a computer tank’s has live bullet to true and observe | Tank movement should appear random, revesing occasionally |
| 7 | If a computer tank can fire, it will circle until it sees an enemy tank **OBJ8** | Create a game with only one tank and one player. Hide from the tank behind an obstacle | Computer tank should be observable moving in circles |
| 8 | If a computer tank can see an enemy but it is far away, it will move forwards towards it **OBJ8** | Create a game with only one tank and one player. Hide from the tank then appear from behind an obstacle but a long way off | Computer tank should move straight to the player tank (firing when it is close enough) |
| 9 | Computer tanks cannot see through walls **OBJ8** | Create a game with only one tank and one player. Hide from the tank behind an obstacle | Computer tank should circle (hunting), and not fire. |
| 10 | If a tank collides with another tank or wall, it will immediately reverse its direction. **OBJ8** | Observe a computer tank near a wall. | If the tank hits a wall, the forward/reverse action should be reversed |
| 11 | Every game has a visible timer, which can be set in the Game Menu **OBJ3** | Click the timer dropdown in the Game Menu | The game timer setting should change |
| 12 | Every game has a visible timer, which can be set in the Game Menu **OBJ3** | Change the timer dropdown in the Game menu, and then click Start Game | Timer on the game screen should reflect the setting made in the Game Menu and count down |
| 13 | Every game ends when the timer reaches zero, if nothing has happened to end the game early. **OBJ3** | Set timer to one minute, create a game with only one player tank and one computer tank. Hide from the computer tank and wait | When the timer reaches zero, the Game results should be shown |
| 14 | The game ends early when all player tanks are destroyed. **OBJ3** | Start a 3 minute Survival game with four player tanks. Get them killed | Game should go to the Game Results screen when last player tank is killed |
| 15 | The game ends early when all player tanks are destroyed. **OBJ3** | Start a 3 minute Last Tank Standing game with four player tanks. Get them killed | Game should go to the Game Results screen when last player tank is killed |
| 16 | When the game ends, the game results are shown **OBJ11** | Let a game run out of time, and start a 1 player game and get killed early | Game Results should be shown |
| 17 | In Last Tank Standing games, a player’s ranking is determined by how many tanks were still alive when the player’s tank died. **OBJ11** | Start a 2 player Last Tank Standing game. Once one tank is destroyed pause the game and count how many live tanks remain | Game result of the first destroyed should be the number of surviving tanks, plus 1 |
| 18 | In Survival games, a player’s ranking is determined by how much they scored **OBJ11** | Start a 2 player game and play normally | Game result should show the highest player score at the top |
| 19 | At the end of Survival Games, if a player has a high score, they are asked to enter their name **OBJ12** | Edit the High Scores file to have very low scores. Play a 1 player game |  |
| 20 | The View High Scores button on the Game menu works **OBJ12** | From the Game Menu, click the View high Scores table and compare with the High Scores text file | Results should match. |
| 21 | Every tank has a difficulty setting, which can be set in the Game Menu **OBJ15** | Start a 3 player game with easy, normal and Hard player settings. Pause the game and observe tank health, speed, and shot speed | The values should differ for each tank, with Easy tanks having the largest values and Hard difficulty tanks having the smallest |
| 22 | The number of player tanks can be set in the Game Menu **OBJ6** | Click the Number of Players dropdown | Options 1 to 4 should be available |
| 23 | The number of enemy tanks can be set in the Game menu **OBJXX** | Start games with 1,2, 3 and 4 players | The corresponding number of players should appear in the game arena |
| 24 | Multiplayer games have split screens, one for each tank **OBJ6** | Start games with 1,2, 3 and 4 players | There are split screen cameras for each of the players |
| 25 | There is a health bar for each player tank on its own screen **OBJ6** | Start a 4-player game | Each camera should have a health bar at the top |
| 26 | A player’s score is visible on its own screen **OBJ6** | Start a 4-player game. Score hits with each tank | The player scores should differ, and be visible on the top right of their own screen |
| 27 | The perspective for a player tank is 3rd person (above and behind). **OBJ4** | Start a 4-player game. Score hits with each tank | Check all views are 3rd person perspective |
| 28 | Game mode can be set in the Game Menu **OBJ7** | Click the Game Mode dropdown | The options Survival and Last Tank Standing should be available. |
| 29 | On Survival games, enemy tanks respawn when they are destroyed **OBJ7** | Start a 1-player, 5 computer tank Survival game. Destroy one enemy tank then pause the game | Computer tank count should still be 5 as the destroyed tank immediately respawns |
| 30 | On Last Tank Standing games, enemy tanks do not respawn. **OBJ7** | Start a 1-player, 5 computer tank Last Tank Standing game. Destroy one enemy tank then pause the game | Computer tank count should be less than 5 as the destroyed tanks are not respawned |
| 31 | When a shell collides with a tank or wall, it will explode **OBJ10** | Start a 1-player, 20 computer tank game. Shoot at a wall, then score a direct hit on a tank | In both cases, there should be an explosion. |
| 32 | When a shell explodes, all tanks nearby are damaged **OBJ10** | Start a 2-player, 1 computer tank game. Move first tank next to a wall, and then shoot with second tank to hit the wall near the first tank | Despite missing the tank directly, the tank should take damage. |
| 33 | In Survival games, when a shell causes damage the damage caused is added to the shooter’s score, if it was a player tank. **OBJ10** | Start a 1-player, 5 tank game. Shoot and cause damage to an enemy tank | Score shown in the player’s view should change. |
| 34 | Clicking the Quit button on the Game Menu ends the game | Click the Quit button | Game shuts down |
| 35 | Clicking the Start button on the Game Menu starts the game | Click the Start button | Game starts |
| 36 | The Game Menu can be accessed at any time by pressing the Escape key, and also closed by pressing the Escape key | Start a game. Press the escape key twice | Game Menu appears on the first press, and disappears on the second press of the Escape key. |

## Post development testing

The post-development tests I will describe are the tests of the final product before the final evaluation is carried out. These tests will evaluate whether the game has met the requirements and will evaluate the game’s usability.

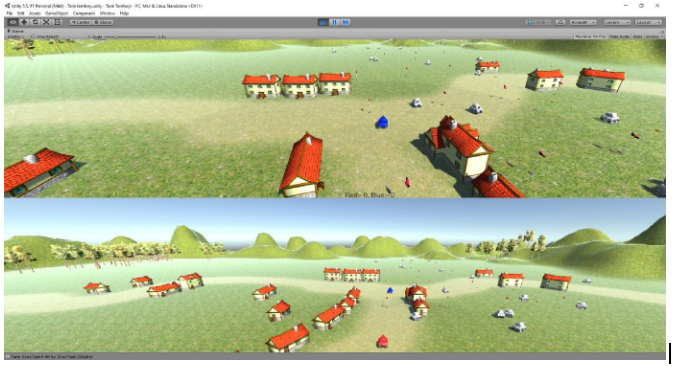
|  |  |
| --- | --- |
| **Test area** | **Test** |
| Game Menu | Do all interactive parts of the game menu work as intended |
| Single player mode | Is the game balanced between damage done and taken, is the speed of the game appropriate |
| Visuals | Are tank explosions correctly placed |
| Gameplay | Do tanks become inactive when Health reaches 0? |
| Multiple player modes | Are the split screens clear? |
| Multiple player modes | Can a player’s score be seen? |
| Multiple player modes | Can a player see their health level? |

# Development and Testing

## Iteration #1 - PROTOTYPE

I had already started developing my project in February, as a part of my practice project. I will use the most recent version as the basis for my game.

Unity has a large number of free assets and game objects in their store, so I have saved development time by taking assets such as the tanks, houses, trees, flags and explosions from the Unity Asset Store. However, all code is my own.



Stakeholder feedback on this version was mixed. Points to be addressed were:

* The battlefield is huge, which makes it difficult for tanks to find each other.
* The stakeholder liked the buildings, trees and landscaping I put in, but they don’t add anything to the game.
* Because the battlefield is huge and each tank is limited to one active shell at a time, missing a shot can be very time consuming while the shell disappears into the distance until it hits something. This was described as ‘very frustrating’
* A direct hit is required on a tank to do damage. In reality, when shells explode they should do damage to everything in a certain radius.
* The AI tanks target the player tanks, but not each other. This is unfair in a Last Tank Standing game.
* A direct hit destroys the enemy – i.e. all tanks have a Health of 1. The stakeholder has already expressed a preference for a fuller health system so some tanks don’t get blown up on the first hit.
* The stakeholder wanted big, medium and small enemy tanks – big ones that could take and deal a lot of damage, and little ones that are fast and hard to hit, but have weak shells and health. (I had already added this in Analysis after doing some research into World of Tanks)
* When tanks are destroyed they disappear. It would be more exciting if there was a better explosion effect.

Other points that the stakeholder didn’t notice but I did, were

* There are no sound effects in the game.
* There is a memory leak in the code – when an explosion occurs ( a Unity Particle system), it isn’t being removed as a game object after the explosion has finished. The game therefore consumes an increasing amount of memory as time goes on, which could crash the PC.

|  |  |
| --- | --- |
| ID | Issue |
| BUG#1 | Explosion particle systems are not being cleaned from memory after being destroyed |

## Iteration #2

### Objectives

On this iteration I will:

* Make a small game arena
* Create the full movement system for one tank only including the ability to fire shells.
* I will create scripts in the Game Manager to spawn enemy tanks
* Write movement routines for the enemy tanks, including firing shells

There won’t be any damage routines, health system, timer, game menu or multiplayer options, but this bare minimum set of procedures will be enough to have a sort of game running that I can show the stakeholder, and get some initial ideas on speed of tank movement and their manoueverability.

At present the code for player tanks is written specifically for the blue (1st) tank. This is for ease of testing. When introducing the multiplayer features and scoring, the code will have to be modularised to be applicable to all player tanks. This will make code maintenance a lot easier.

### Test plan

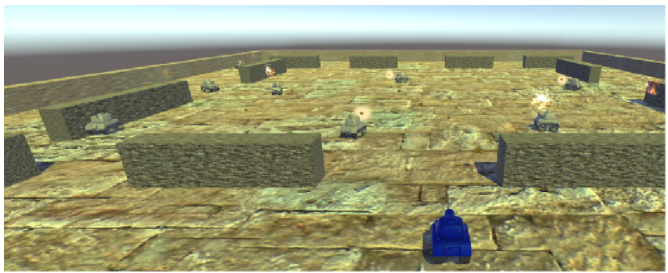
Tests for the above features will be:

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Statement | Test | Expected result |
| 1 | Player tanks move forward, back, left and right in response to key presses | For each tank, check the movement keypresses in the Design section are correct. | Movement in line with Design document expectations |
| 2 | Player tanks move appropriately when a combination of move keypresses are made. | For each tank, press combinations of Forward/back, and left/right | Tanks should move in circles |
| 3 | Player tanks fire in response to it’s shooting keypress | For each tank, press the Fire key | Tank should shoot |
| *(the above tests are only partial tests because they can only check for the 1st tank. We will retest when multiplayer features are added)* | | | |
| 4 | The game arena is a hard obstacle, which stops shells and tanks | Run a player tank into all the walls | The tank should be blocked by the wall |
| 5 | Every tank can only have one live shell at a time. | With a player tank, fire and then attempt to fire again | No second shell should be fired |
| 6 | If a computer tank cannot fire, it will move evasively | Set a computer tank’s has live bullet to true and observe | Tank movement should appear random, revesing occasionally |
| 7 | If a computer tank can fire, it will circle until it sees an enemy tank | Create a game with only one tank and one player. Hide from the tank behind an obstacle | Computer tank should be observable moving in circles |
| 8 | If a computer tank can see an enemy but it is far away, it will move forwards towards it | Create a game with only one tank and one player. Hide from the tank then appear from behind an obstacle but a long way off | Computer tank should move straight to the player tank (firing when it is close enough) |
| 9 | Computer tanks cannot see through walls | Create a game with only one tank and one player. Hide from the tank behind an obstacle | Computer tank should circle (hunting), and not fire. |
| 10 | If a tank collides with another tank or wall, it will immediately reverse its direction. | Observe a computer tank near a wall. | If the tank hits a wall, the forward/reverse action should be reversed |
| 27 | The perspective for a player tank is 3rd person (above and behind). | Start a 4-player game. Score hits with each tank | Check all views are 3rd person perspective |

### Game Arena

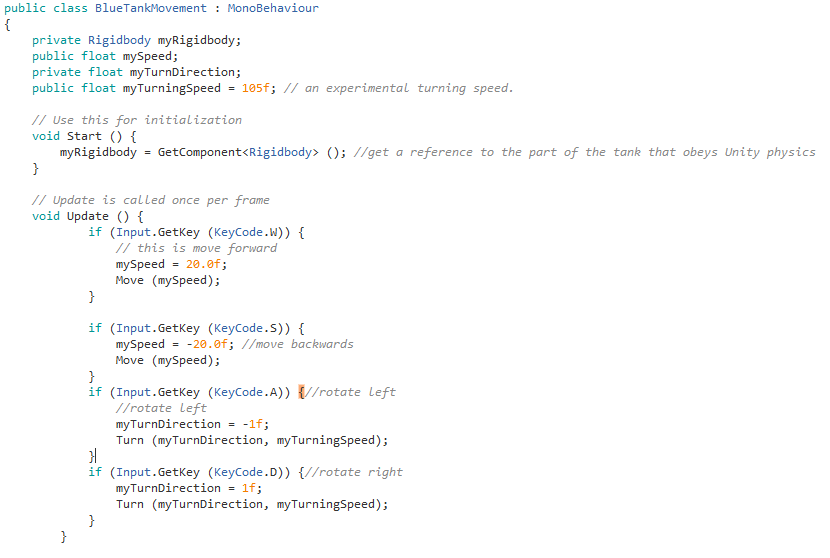
This is now a lot smaller, and contained within a box of walls, so there is no chance of a player having to wait to fire again while their shell disappears in to the far distance.

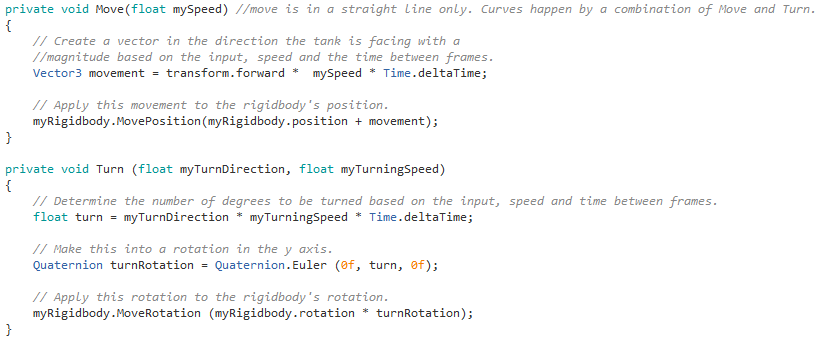
This battlefield is much simpler than the one I had previously, and hence it is easier to traverse and find other tanks. This makes the gameplay more fun.



### Player tank movement

This is the script for the blue player tank movement





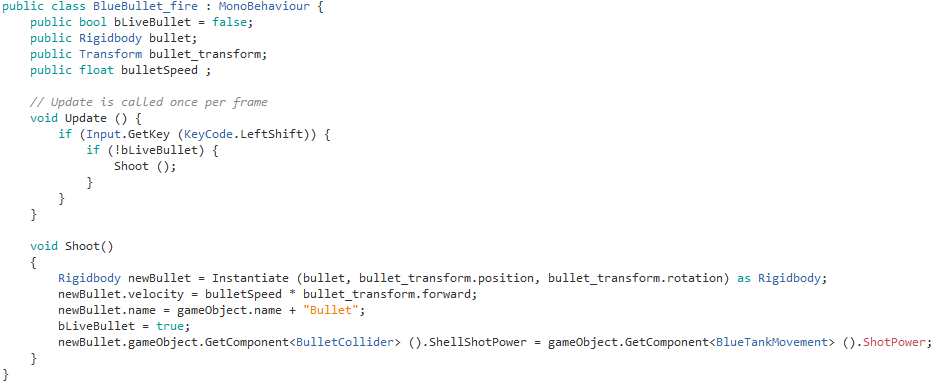
This section of the code allows the player to move their tank forwards and backwards, and to rotate their tank clockwise and anticlockwise. Rotating the tank rotates the game view, as there is a camera object positioned behind each player tank that is a child of the tank, and hence rotates with the tank.

When the game starts and the object is instantiated, the tank’s rigidbody is fetched using its reference. Then, once per frame the program checks for any keyboard input. I have used multiple if statements to specify the movement keys, and to execute a certain procedure based on which key has been pressed. There isn’t branching, to allow the player to move and rotate their tank simultaneously. However, this means that the player can press both the left and right rotate buttons, or the forward and backwards buttons at the same time which cancels the movement or rotation of the tank.

### Player shooting

Player shooting is straightforward. The Update statement in the script checks to see if the key bound to shooting has been pressed. It checks a Boolean value, bLiveBullet, to make sure the player doesn’t already have a live shell. If it is ok to fire, then in the method Shoot(), a shell is created pointing in the same direction as the barrel of the tank, and bLiveBullet is set to true so the player cannot fire again until this shell is destroyed.

On being instantiated, the shell is given the name of the tank that fired it, plus “Bullet” on the end. This is so that later we will be able to tell who shot who. This is the only connection between tanks and their shells. The shell is a prefab shape that I took from the Unity store and bound it at design time in the Unity development environment.



### Spawning enemy tanks

This is the script for creating enemy tanks. On the Game arena, I will have 20 locations marked on the arena floor using Transform objects (these are just locations in 3d space with 3d direction information included). The script instantiates generic tanks (different tank types will come later) in these locations, and gives them default attributes for health, speed and shell power.

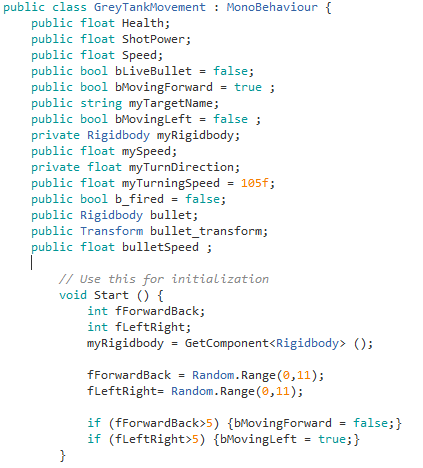


For the game mode ‘Survival’, any destroyed tank is immediately replaced with a new one. The script below will achieve this, though I can’t test this yet as I don’t have damage or destruction of tanks built into the system.

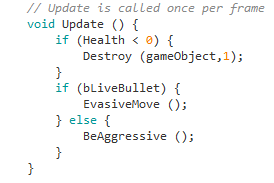


### Computer tank movement logic and shooting

First, I set up the tank moving in a random direction

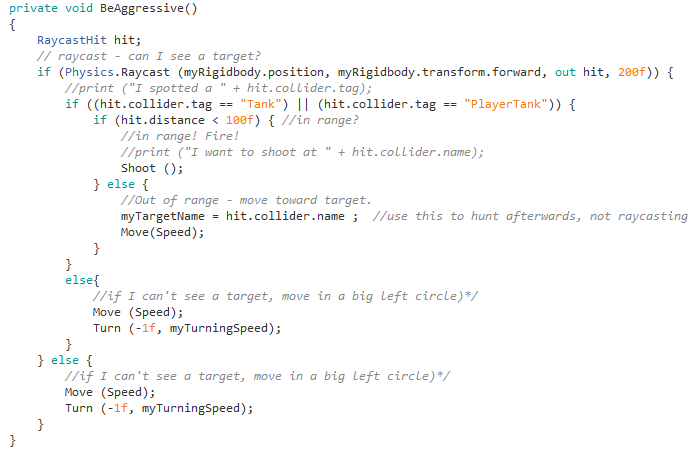


The core of the computer tank’s logic is whether it can fire yet or not. Update() is called once per frame.



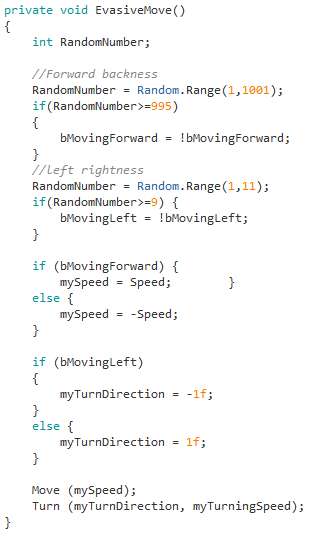
i.e. if the tank can fire, it will BeAggressive – hunt other tanks and shoot at them. Otherwise it will be evasive and move randomly.

being aggressive …



Here, Shoot() is exactly the same code as used by the player tanks.

If the tank has already fired, it needs to be evasive until it can fire again. The procedure EvasiveMove() defines the behaviour of the AI tank after it has fired a bullet, and while it is waiting to fire another. There is a 1/200 chance of the tank reversing, and there is a 3/11 chance of the tank rotating towards the right. These figures were arrived at by experimentation. Since the procedure is called every frame, a 1/200 chance of reversing direction means on average the tank reverses every 3 or 4 seconds, just enough to make it frustrating for a player trying to chase and hit it.



Move() and Turn() are exactly the same as for player tanks.

### Results of Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Statement | Test | Expected result | Result |
| 1 | Player tanks move forward, back, left and right in response to key presses | For each tank, check the movement keypresses in the Design section are correct. | Movement in line with Design document expectations | Pass |
| 2 | Player tanks move appropriately when a combination of move keypresses are made. | For each tank, press combinations of Forward/back, and left/right | Tanks should move in circles | Pass |
| 3 | Player tanks fire in response to it’s shooting keypress | For each tank, press the Fire key | Tank should shoot | Pass |
| *(the above tests are only partial tests because they can only check for the 1st tank. We will retest when multiplayer features are added)* | | | |  |
| 4 | The game arena is a hard obstacle, which stops shells and tanks | Run a player tank into all the walls | The tank should be blocked by the wall | Pass |
| 5 | Every tank can only have one live shell at a time. | With a player tank, fire and then attempt to fire again | No second shell should be fired | Pass |
| 6 | If a computer tank cannot fire, it will move evasively | Set a computer tank’s has live bullet to true and observe | Tank movement should appear random, revesing occasionally | Pass |
| 7 | If a computer tank can fire, it will circle until it sees an enemy tank | Create a game with only one tank and one player. Hide from the tank behind an obstacle | Computer tank should be observable moving in circles | Pass |
| 8 | If a computer tank can see an enemy but it is far away, it will move forwards towards it | Create a game with only one tank and one player. Hide from the tank then appear from behind an obstacle but a long way off | Computer tank should move straight to the player tank (firing when it is close enough) | Fail – the AI tank starts moving towards the enemy, but loses interest as soon as it is not directly in front |
| 9 | Computer tanks cannot see through walls | Create a game with only one tank and one player. Hide from the tank behind an obstacle | Computer tank should circle (hunting), and not fire. | Pass |
| 10 | If a tank collides with another tank or wall, it will immediately reverse its direction. | Observe a computer tank near a wall. | If the tank hits a wall, the forward/reverse action should be reversed | Fail – forgot to implement this |
| 27 | The perspective for a player tank is 3rd person (above and behind). | Start a 4-player game. Score hits with each tank | Check all views are 3rd person perspective | Pass |

So I have 2 items to fix –

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | If a computer tank can see an enemy but it is far away, it will move forwards towards it | Create a game with only one tank and one player. Hide from the tank then appear from behind an obstacle but a long way off | Computer tank should move straight to the player tank (firing when it is close enough) |
| 10 | If a tank collides with another tank or wall, it will immediately reverse its direction. | Observe a computer tank near a wall. | If the tank hits a wall, the forward/reverse action should be reversed |

Neither of these appear to be critical problems, so I will deal with them at an appropriate point in development, instead of making them top priority.

### Stakeholder feedback

The stakeholder decided the tank movement speed and rotation speed. He did not think the player tank shells moved quickly enough, and was a bit surprised that there was only one player on the screen, as he’d seen a previous version where there were two. I told him that this was early in the development process and this feature would be added later.

## Iteration #3

Now that I have the framework of a game, I will start to add more features that the stakeholder wanted.

### Objectives

In this development cycle I will:

* Add small and big computer tanks to the existing computer tank.
* Have exploding shells cause damage.
* Fix the memory leak:

|  |  |
| --- | --- |
| ID | Issue |
| BUG#1 | Explosion particle systems are not being cleaned from memory after being destroyed |

### Test Plan

Tests for the above features will be

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Statement | Test | Expected result |
| 31 | When a shell collides with a tank or wall, it will explode | Start a 1-player, 20 computer tank game. Shoot at a wall, then score a direct hit on a tank | In both cases, there should be an explosion. |
| 32 | When a shell explodes, all tanks nearby are damaged | Start a 2-player, 1 computer tank game. Move first tank next to a wall, and then shoot with second tank to hit the wall near the first tank | Despite missing the tank directly, the tank should take damage. |
| 33 | In Survival games, when a shell causes damage the damage caused is added to the shooter’s score, if it was a player tank. | Start a 1-player, 5 tank game. Shoot and cause damage to an enemy tank | Score shown in the player’s view should change. |
| New Test (missed in Design) | | | |
| 37 | The game populates with 3 sizes of computer tank, with varying speed and health | Start a 1 player game with 20 computer tanks. | There should be a mix of different sizes of tank. |
| 38 | The game populates with 3 sizes of computer tank, with varying speed and health | Start a 1 player game with 20 computer tanks. Pause the game and inspect different tank’s health and speed parameters | There should be a mix of different sizes of tank. |

### Different Tank Sizes

My stakeholder would like to see some variety in the types of enemies he fights, so instead of 20 identical enemy tanks on the battlefield, the player(s) will fight big tanks which can deal a lot of damage and take a lot of damage without being destroyed, but are slow-moving, medium tanks, and small tanks which can move quickly, but have a low damage threshold and deal a lot less damage than other tanks.

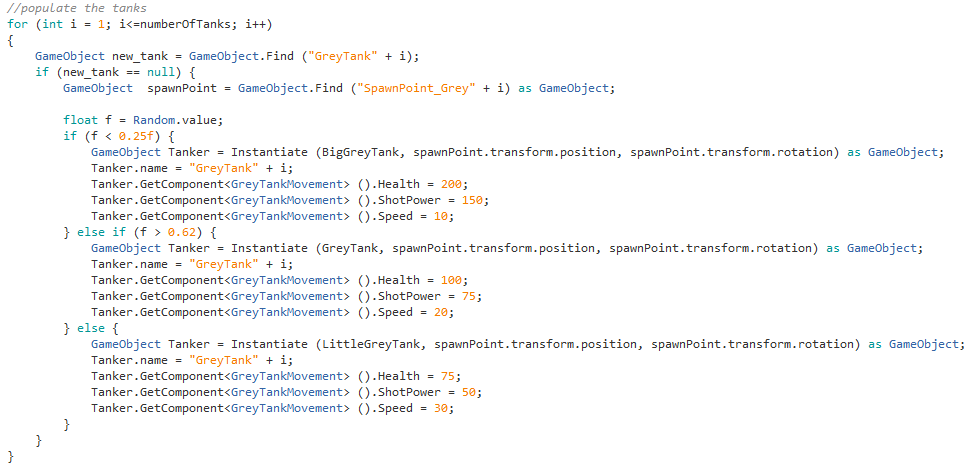
Unity has two different types of objects: scene objects and prefabs. Prefabs are objects with preconfigured behaviours (scripts) and attributes, and are available at runtime. Prefabs can be called by instantiation. My basic AI tank prefab had already been configured, so I have duplicated the prefab twice, and adjusted the sizes of the duplicates.

When i adjusted heights of tanks, i found that the bullets passed over the top of the smaller tanks. To solve this problem, I decided to only adjust the widths and lengths of the tanks.

I have created three reference variables for the different types of tanks in the Game manager:



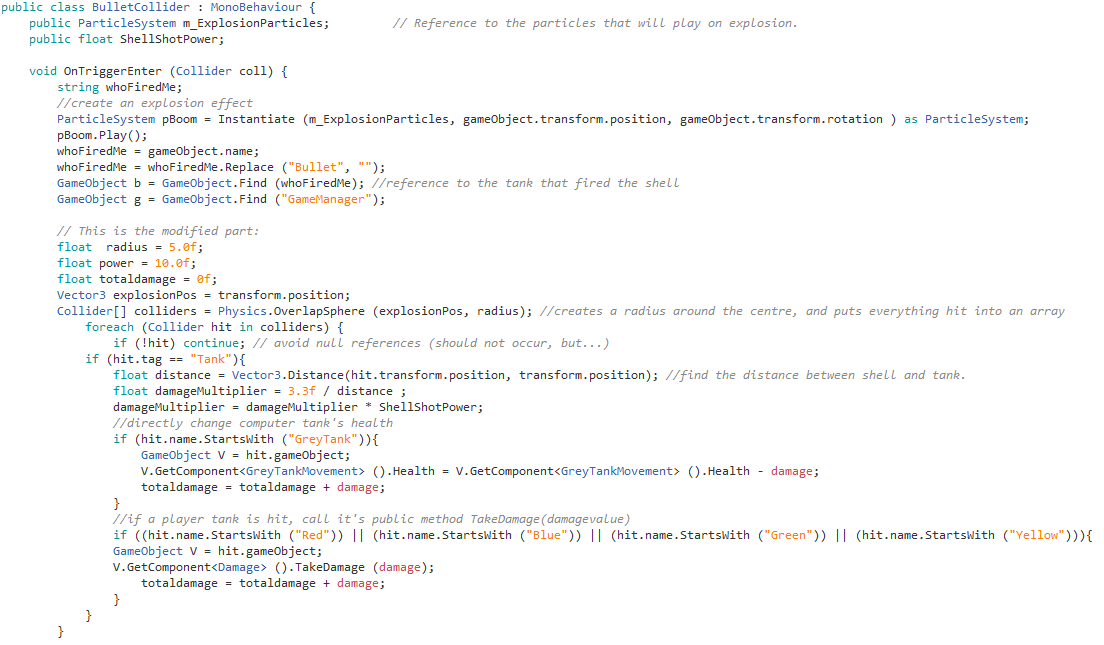
I have also altered the procedure that handles spawning AI tanks in the Game Manager. It now spawns random amounts of the different types of tanks, and sets the speed, health and firepower according to the type of tank.



### Area effect damage for shells

While testing the original version, I noticed that a direct hit is required on a tank to do damage. It is easy to miss a tank this way, and in reality, when shells explode they should do damage to everything in a certain radius.

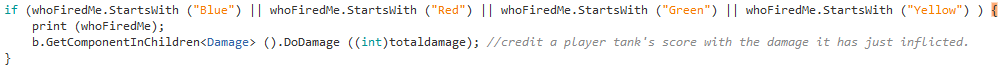
To fix this issue, I have written the following code in the OnTriggerEnter(Collider coll) procedure. This is called whenever a shell’s collider object strikes another collider object in the game.



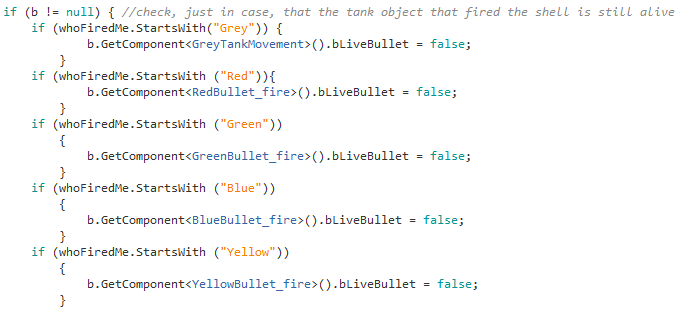
This creates a radius around the shell and detects all tanks that it intersects. It then calculates damage based on distance from the shell to the nearest point on the tank.

TakeDamage is a method common to all player tanks, and it decreases that tank’s health.

If a player tank did the shooting, then they need to have their score increased by the amount of damage they did. DoDamage is a common method for all player tanks:



Finally, whoever fired the shell can fire again. This is a little messy and not object-oriented, because at present I have a specific firing script for each player tank. This will be fixed in a later iteration.



### Fix the memory leak

Finally, I have fixed the memory leak caused by the explosion particle effects not being deleted.

In Unity, explosions are handled as ParticleSystem objects, which have a limited lifespan. The ParticleSystem starts, blooms and shrinks. When the particleSystem animation has finished, code is needed to destroy the object.

When I tested the game, I found that the code I had wrote to destroy explosion ParticleSystems was not working. I searched for a solution to this problem on various Unity Forums, but all of the proposed solutions do not work in my game.

This may be because of the hierarchy I set up for explosion effects:

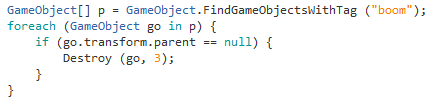
Tank object is the parent object

Shell object is the child of the tank object

Explosion ParticleSystem is the child of the shell object.

In the original version of the code I had written, on a collision, an explosion particle effect is created, and plays on top of the shell. The shell is then destroyed before the explosion. I believe I could not destroy the explosion ParticleSystem because it no longer had a parent, so it couldn’t be destroyed using the parent reference.

I needed to use a different approach to destroy the explosion ParticleSystems without using its parent reference. In the GameManager object, I wrote a few lines of code, as shown below:



The GameManager finds every explosion object in the scene, and stores their references in the array p. It then checks if each explosion object has a parent. If the explosion is parentless, it is destroyed after 8 seconds, which is the amount of time the explosion takes to start, bloom, and shrink.

This method works, and the memory leak has been fixed.

### Results of Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Statement | Test | Expected result | Result |
| 31 | When a shell collides with a tank or wall, it will explode | Start a 1-player, 20 computer tank game. Shoot at a wall, then score a direct hit on a tank | In both cases, there should be an explosion. | Pass |
| 32 | When a shell explodes, all tanks nearby are damaged | Start a 2-player, 1 computer tank game. Move first tank next to a wall, and then shoot with second tank to hit the wall near the first tank | Despite missing the tank directly, the tank should take damage. | Pass |
| 33 | In Survival games, when a shell causes damage the damage caused is added to the shooter’s score, if it was a player tank. | Start a 1-player, 5 tank game. Shoot and cause damage to an enemy tank | Score shown in the player’s view should change. | Pass – though the score is recorded for both Game modes |
| 37 | The game populates with 3 sizes of computer tank, with varying speed and health | Start a 1 player game with 20 computer tanks. | There should be a mix of different sizes of tank. | Pass |
| 38 | The game populates with 3 sizes of computer tank, with varying speed and health | Start a 1 player game with 20 computer tanks. Pause the game and inspect different tank’s health and speed parameters | There should be a mix of different sizes of tank. | Pass |

### Stakeholder feedback

Stakeholder feedback was very positive. The stakeholder accepted that the multiscreen game was still a way away, but enjoyed the feel of the game in its limited form. I was given a broad rule of thumb for shell strength and tank health – the stakeholder felt that three direct hits should be the right amount for a medium tank to destroy another medium tank. Accordingly I adjusted a damage multiplier in the damage routines to get this working.

## Iteration #4

At this point I have a game where one player can drive around, shoot at other tanks and cause damage with radius-effect shells. What remains is to tie things up by adding the final features.

### Objectives

I will do the following on this development cycle:

* Make the game multiplayer, with split screens
* Put in health bars for each player.
* Put in the Menu screen and load the user’s settings back into the Game Manager
* Have the Game Manager detect when the game has ended
* Show game results at the end of a game.
* Put in a High Score table
* Fix outstanding errors –

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | If a computer tank can see an enemy but it is far away, it will move forwards towards it | Create a game with only one tank and one player. Hide from the tank then appear from behind an obstacle but a long way off | Computer tank should move straight to the player tank (firing when it is close enough) |
| 10 | If a tank collides with another tank or wall, it will immediately reverse its direction. | Observe a computer tank near a wall. | If the tank hits a wall, the forward/reverse action should be reversed |

### Test Plan

The following tests will be executed

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Statement | Test | Expected result |
| 1 | Player tanks move forward, back, left and right in response to key presses | For each tank, check the movement keypresses in the Design section are correct. | Movement in line with Design document expectations |
| 2 | Player tanks move appropriately when a combination of move keypresses are made. | For each tank, press combinations of Forward/back, and left/right | Tanks should move in circles |
| 3 | Player tanks fire in response to it’s shooting keypress | For each tank, press the Fire key | Tank should shoot |
| 11 | Every game has a visible timer, which can be set in the Game Menu | Click the timer dropdown in the Game Menu | The game timer setting should change |
| 12 | Every game has a visible timer, which can be set in the Game Menu | Change the timer dropdown in the Game menu, and then click Start Game | Timer on the game screen should reflect the setting made in the Game Menu and count down |
| 13 | Every game ends when the timer reaches zero, if nothing has happened to end the game early. | Set timer to one minute, create a game with only one player tank and one computer tank. Hide from the computer tank and wait | When the timer reaches zero, the Game results should be shown |
| 14 | The game ends early when all player tanks are destroyed. | Start a 3 minute Survival game with four player tanks. Get them killed | Game should go to the Game Results screen when last player tank is killed |
| 15 | The game ends early when all player tanks are destroyed. | Start a 3 minute Last Tank Standing game with four player tanks. Get them killed | Game should go to the Game Results screen when last player tank is killed |
| 16 | When the game ends, the game results are shown | Let a game run out of time, and start a 1 player game and get killed early | Game Results should be shown |
| 17 | In Last Tank Standing games, a player’s ranking is determined by how many tanks were still alive when the player’s tank died. | Start a 2 player Last Tank Standing game. Once one tank is destroyed pause the game and count how many live tanks remain | Game result of the first destroyed should be the number of surviving tanks, plus 1 |
| 18 | In Survival games, a player’s ranking is determined by how much they scored | Start a 2 player game and play normally | Game result should show the highest player score at the top |
| 19 | At the end of Survival Games, if a player has a high score, they are asked to enter their name | Edit the High Scores file to have very low scores. Play a 1 player game |  |
| 20 | The View High Scores button on the Game menu works | From the Game Menu, click the View high Scores table and compare with the High Scores text file | Results should match. |
| 21 | Every tank has a difficulty setting, which can be set in the Game Menu | Start a 3 player game with easy, normal and Hard player settings. Pause the game and observe tank health, speed, and shot speed | The values should differ for each tank, with Easy tanks having the largest values and Hard difficulty tanks having the smallest |
| 22 | The number of player tanks can be set in the Game Menu | Click the Number of Players dropdown | Options 1 to 4 should be available |
| 23 | The number of enemy tanks can be set in the Game menu | Start games with 1,2, 3 and 4 players | The corresponding number of players should appear in the game arena |
| 24 | Multiplayer games have split screens, one for each tank | Start games with 1,2, 3 and 4 players | There are split screen cameras for each of the players |
| 25 | There is a health bar for each player tank on its own screen | Start a 4-player game | Each camera should have a health bar at the top |
| 26 | A player’s score is visible on its own screen | Start a 4-player game. Score hits with each tank | The player scores should differ, and be visible on the top right of their own screen |
| 28 | Game mode can be set in the Game Menu | Click the Game Mode dropdown | The options Survival and Last Tank Standing should be available. |
| 29 | On Survival games, enemy tanks respawn when they are destroyed | Start a 1-player, 5 computer tank Survival game. Destroy one enemy tank then pause the game | Computer tank count should still be 5 as the destroyed tank immediately respawns |
| 30 | On Last Tank Standing games, enemy tanks do not respawn. | Start a 1-player, 5 computer tank Last Tank Standing game. Destroy one enemy tank then pause the game | Computer tank count should be less than 5 as the destroyed tanks are not respawned |
| 34 | Clicking the Quit button on the Game Menu ends the game | Click the Quit button | Game shuts down |
| 35 | Clicking the Start button on the Game Menu starts the game | Click the Start button | Game starts |
| 36 | The Game Menu can be accessed at any time by pressing the Escape key, and also closed by pressing the Escape key | Start a game. Press the escape key twice | Game Menu appears on the first press, and disappears on the second press of the Escape key. |

### Game Menu

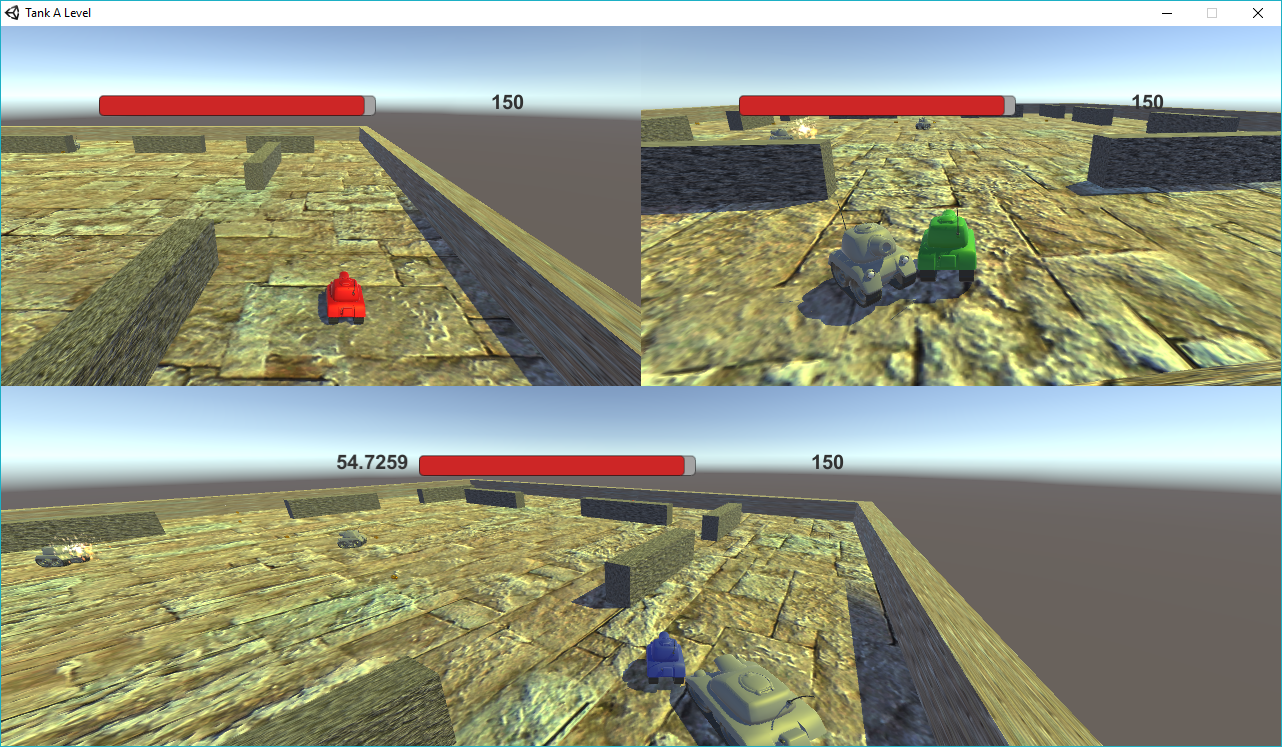
The game menu has been implemented in line with the Design, but without the difficulty dropdowns. This is done by using a Canvas object and placing labels and dropdowns onto it. The dropdowns are bound to routines in the code that change game settings, or show other screens.





### Split screens for multiplayer games

This was more difficult. Each player tank object has a camera attached to it (set behind and above, to give the 3rd person perspective). Where on the screen a player’s camera is placed, and how wide/tall it is, depends on the number of players.



A video has been made showing the menu and split screens in operation.

### Results of Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Statement | Test | Expected result | Result |
| 1 | Player tanks move forward, back, left and right in response to key presses | For each tank, check the movement keypresses in the Design section are correct. | Movement in line with Design document expectations | Pass |
| 2 | Player tanks move appropriately when a combination of move keypresses are made. | For each tank, press combinations of Forward/back, and left/right | Tanks should move in circles | Pass |
| 3 | Player tanks fire in response to it’s shooting keypress | For each tank, press the Fire key | Tank should shoot | Pass |
| 11 | Every game has a visible timer, which can be set in the Game Menu | Click the timer dropdown in the Game Menu | The game timer setting should change | Pass |
| 12 | Every game has a visible timer, which can be set in the Game Menu | Change the timer dropdown in the Game menu, and then click Start Game | Timer on the game screen should reflect the setting made in the Game Menu and count down | Pass |
| 13 | Every game ends when the timer reaches zero, if nothing has happened to end the game early. | Set timer to one minute, create a game with only one player tank and one computer tank. Hide from the computer tank and wait | When the timer reaches zero, the Game results should be shown | Pass |
| 14 | The game ends early when all player tanks are destroyed. | Start a 3 minute Survival game with four player tanks. Get them killed | Game should go to the Game Results screen when last player tank is killed | Pass |
| 15 | The game ends early when all player tanks are destroyed. | Start a 3 minute Last Tank Standing game with four player tanks. Get them killed | Game should go to the Game Results screen when last player tank is killed | Pass |
| 16 | When the game ends, the game results are shown | Let a game run out of time, and start a 1 player game and get killed early | Game Results should be shown | Pass |
| 17 | In Last Tank Standing games, a player’s ranking is determined by how many tanks were still alive when the player’s tank died. | Start a 2 player Last Tank Standing game. Once one tank is destroyed pause the game and count how many live tanks remain | Game result of the first destroyed should be the number of surviving tanks, plus 1 | Pass |
| 18 | In Survival games, a player’s ranking is determined by how much they scored | Start a 2 player game and play normally | Game result should show the highest player score at the top | Pass |
| 19 | At the end of Survival Games, if a player has a high score, they are asked to enter their name | Edit the High Scores file to have very low scores. Play a 1 player game |  | Fail – feature not built |
| 20 | The View High Scores button on the Game menu works | From the Game Menu, click the View high Scores table and compare with the High Scores text file | Results should match. | Pass |
| 21 | Every tank has a difficulty setting, which can be set in the Game Menu | Start a 3 player game with easy, normal and Hard player settings. Pause the game and observe tank health, speed, and shot speed | The values should differ for each tank, with Easy tanks having the largest values and Hard difficulty tanks having the smallest | Pass |
| 22 | The number of player tanks can be set in the Game Menu | Click the Number of Players dropdown | Options 1 to 4 should be available | Pass |
| 23 | The number of enemy tanks can be set in the Game menu | Start games with 1,2, 3 and 4 players | The corresponding number of players should appear in the game arena | Pass |
| 24 | Multiplayer games have split screens, one for each tank | Start games with 1,2, 3 and 4 players | There are split screen cameras for each of the players | Pass |
| 25 | There is a health bar for each player tank on its own screen | Start a 4-player game | Each camera should have a health bar at the top | Pass |
| 26 | A player’s score is visible on its own screen | Start a 4-player game. Score hits with each tank | The player scores should differ, and be visible on the top right of their own screen | Pass |
| 28 | Game mode can be set in the Game Menu | Click the Game Mode dropdown | The options Survival and Last Tank Standing should be available. | Pass |
| 29 | On Survival games, enemy tanks respawn when they are destroyed | Start a 1-player, 5 computer tank Survival game. Destroy one enemy tank then pause the game | Computer tank count should still be 5 as the destroyed tank immediately respawns | Pass |
| 30 | On Last Tank Standing games, enemy tanks do not respawn. | Start a 1-player, 5 computer tank Last Tank Standing game. Destroy one enemy tank then pause the game | Computer tank count should be less than 5 as the destroyed tanks are not respawned | Pass |
| 34 | Clicking the Quit button on the Game Menu ends the game | Click the Quit button | Game shuts down | Pass |
| 35 | Clicking the Start button on the Game Menu starts the game | Click the Start button | Game starts | Pass |
| 36 | The Game Menu can be accessed at any time by pressing the Escape key, and also closed by pressing the Escape key | Start a game. Press the escape key twice | Game Menu appears on the first press, and disappears on the second press of the Escape key. | Pass |

### Stakeholder feedback

As this was the final iteration, stakeholder feedback has been presented in the Evaluation section, below.

# Evaluation

## Requirement Check

I have passed all but two requirements.

|  |  |  |
| --- | --- | --- |
| **Ref** | **High Level Objective** | **Pass/Fail** |
| OBJ1 | The game must be playable on a Windows PC. | Fully Met |
| OBJ2 | The game area should be small, with obstacles. | Fully Met |
| OBJ3 | There should be a time limit on the games | Fully Met |
| OBJ4 | The perspective should be 3rd person (“above and behind”) for the players. | Fully Met |
| OBJ5 | Tank movement should be as per two-track vehicle movement – i.e. tanks should be able to move forward and backwards, rotate in either direction, and move in an arc. They should not be able to move diagonally or sideways. | Fully Met |
| OBJ6 | There should be the option for 1 to 4 players in the same game using split screens,  and all player actions coming from the same keyboard. | Fully Met |
| OBJ7 | There should be two game modes. These are Survival, where the players accumulate scores for doing damage to the other tanks and try to last to the time limit, and Last Tank Standing, where the objective is to be the sole remaining tank | Fully Met |
| OBJ8 | Computer-controlled tanks should operate on line of sight, and be hard to hit. | Partially met |
| OBJ9 | All tanks can fire shells | Fully Met |
| OBJ10 | Shells explode on collision with obstacles and other tanks, causing damage to every tank within the explosion radius. | Fully Met |
| OBJ11 | For Survival game mode, there should be a scoring system for doing damage to other tanks. | Fully Met |
| OBJ12 | There should be a high score table for the best Survival scores. | Fail |
| OBJ13 | Tanks should be able to survive more than one direct hit. | Fully Met |
| OBJ14 | There should be a Difficulty setting for Player Tanks | Fail |

I do not have a high score table for survival scores, and it was not possible in the time available to implement the additional requirement I added in – that there should be difficulty settings for players. Also feedback from the stakeholder was that while the enemy tanks were very evasive when avoiding fire, their movement pattern when hunting was predictable and this could be improved. This is why I have put a Partially Met for OBJ8. (The tanks were programmed to do a big circle when hunting and this is what they do, so the mistake is more in the design of the tank logic than any error in programming.)

## Feedback from stakeholder - Jeremy Erwen

Jeremy has agreed that the game meets most of the requirements, and has stated that the game was easy to understand and use. He has suggested that in the future, I should make the visuals more cartoonish. (I had a cartoon style explosion particle system, but changed it later for a more realistic oil fire explosion. I should have kept the first one).

## Level of Success of usability features

Jeremy found the drop down menus easy to use, and could easily understand the controls for each player. However, on the results screen for the Last Tank Standing Mode, he was confused as to the number to the left of his player name, which I explained to him was his ranking. He suggested that in future development, I should put a “Rank” title above the rankings.

Jeremy also did not know how to input his name in the name field when recording a new high score, as there was no blinking cursor to indicate an input field.

As Idris has a short attention span, he did not properly read the keyboard controls for his tank on the main menu screen. However, once I told him the controls, he caught on quickly and could play the game as normal. As with Jeremy, I had to show him how to input his name when recording a new high score.

## Issues, Maintenance Issues and Limitations of solution

Some of the limitations that my stakeholders and I have come across are:

* In 1 player mode, the health bar, score and timer are positioned in the middle of the screen.
* When a player tank runs out of HP, its movement and firing functions are disabled, instead of the tank disappearing.
* The game’s visuals look too simple, and not as cartoon like as was intended.
* The name input box for recording a new high score does not show a blinking cursor, and there is no instruction telling the user to record their name, so a user would not know how to input their name.
* The noise made by multiple shell explosions happening at the same time when two tanks are in close range sounds jarring.
* The game’s performance can lag on lower-specification hardware, especially when there are many AI tanks in one game, shooting within close range of each other.
* In Last Tank Standing Mode, on the game over screen, the ranking of each player is shown but there is nothing that states that they are the rankings. Hence, it is not obvious what they are.

## How partially met/unmet criteria and limitations could be addressed by further development

By further development, I would be able to implement multiple levels of difficulty for the player tanks by adjusting the speed of the tank, the firepower of their shells and the speed their shells would travel at.

I would also adjust the positions of the health bar, score and timer for the 1 player mode so they are at the top of the screen. Additionally, I would replace the text and buttons in the User Interface into more cartoon-like assets.

I would investigate further into the issue of the camera for each player disappearing when their tank is destroyed in multiplayer mode. At the moment, I believe it is due to the fact that the camera for each player tank is a child of the tank itself. Once I have fixed this issue, I would adjust the code so that when a tank has ran out of HP, it is destroyed.

At the moment, I am not sure how I would fix the performance, or sound issues, but with further development time, I would be able to investigate into these problems and attempt to fix the issues.

# ALL TANK TERRITORY CODE

***BlueBullet\_fire***

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BlueBullet\_fire : MonoBehaviour {

public AudioClip shotFire;

public AudioSource FiringSource;

public bool bLiveBullet = false;

public Rigidbody bullet;

public Transform bullet\_transform;

public float bulletSpeed ;

// Use this for initialization

void Start () {

FiringSource.clip = shotFire;

}

// Update is called once per frame

void Update () {

if (gameObject.GetComponentInChildren<Damage> ().bCanScore == true) {

if (Input.GetKey (KeyCode.LeftShift)) {

if (!bLiveBullet) {

Shoot ();

}

}

}

if (Input.GetKey(KeyCode.I)) {

print ("Bullet\_transform position: " + bullet\_transform.position);

}

}

void Shoot()

{

Rigidbody newBullet = Instantiate (bullet, bullet\_transform.position, bullet\_transform.rotation) as Rigidbody;

newBullet.velocity = bulletSpeed \* bullet\_transform.forward;

newBullet.name = gameObject.name + "Bullet";

bLiveBullet = true;

newBullet.gameObject.GetComponent<BulletCollider> ().ShellShotPower = gameObject.GetComponent<BlueTankMovement> ().ShotPower;

FiringSource.Play ();

}

}

***BlueTankMovement***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class BlueTankMovement : MonoBehaviour   
{  
    public float ShotPower;  
    public float Speed;  
    public int FlagCount = 0;  
    private Rigidbody myRigidbody;  
    public float mySpeed;  
    private float myTurnDirection;  
    public float myTurningSpeed = 105f;  
  
    *// Use this for initialization*  
    void Start () {  
        myRigidbody = GetComponent<Rigidbody> ();  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanMove == true ) {  
            if (Input.GetKey (KeyCode.W)) {  
                *// this is where I move forward*  
                mySpeed = 20.0f;  
                Move (mySpeed);  
            }  
  
            if (Input.GetKey (KeyCode.S)) {  
                mySpeed = -20.0f;  
                Move (mySpeed);  
            }  
            if (Input.GetKey (KeyCode.A)) {  
                *//rotate left*  
                myTurnDirection = -1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            if (Input.GetKey (KeyCode.D)) {  
                *//rotate right*  
                myTurnDirection = 1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            if (Input.GetKey (KeyCode.V)) {  
                *//Test for Slider Bar*  
                print (gameObject.GetComponentInChildren<Slider> ().normalizedValue);  
                gameObject.GetComponentInChildren<Slider> ().value = 0.75f;  
            }  
  
            */\*if (Input.GetKey (KeyCode.R)) {*  
*//test raycast*  
*Debug.DrawRay(myRigidbody.position ,myRigidbody.transform.forward \* 100f, Color.red);*  
*print (myRigidbody.name) ;*  
*print (myRigidbody.transform.position);*  
*}\*/*  
  
        }  
    }  
    private void Move(float mySpeed)  
    {  
        *// Create a vector in the direction the tank is facing with a magnitude based on the input, speed and the time between frames.*  
        Vector3 movement = transform.forward \*  mySpeed \* Time.deltaTime;  
  
        *// Apply this movement to the rigidbody's position.*  
        myRigidbody.MovePosition(myRigidbody.position + movement);  
    }  
  
  
    void OnCollisionEnter(Collision coll) {  
        *//print ("collided with " + collision.collider.ToString ());*  
        *//print ("collided with " + collision.collider.tag);*  
        GameObject g;  
        *//print (coll.gameObject.name);*  
        if (coll.gameObject.tag == "FlagPole") {  
            FlagCount++;  
            print ("i hit " );  
            Destroy (coll.transform.parent.gameObject );  
            g = GameObject.Find ("GameManager");  
            g.GetComponent<GameManager>().BluScore =g.GetComponent<GameManager>().BluScore + 3;   
            *//Destroy (collision.collider, 0);*  
        }  
    }          
  
  
    private void Turn (float myTurnDirection, float myTurningSpeed)  
    {  
        *// Determine the number of degrees to be turned based on the input, speed and time between frames.*  
        float turn = myTurnDirection \* myTurningSpeed \* Time.deltaTime;  
  
        *// Make this into a rotation in the y axis.*  
        Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);  
  
        *// Apply this rotation to the rigidbody's rotation.*  
        myRigidbody.MoveRotation (myRigidbody.rotation \* turnRotation);  
    }  
  
  
  
  
}

***BulletCollider***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class BulletCollider : MonoBehaviour {  
    public LayerMask m\_TankMask;                        *// Used to filter what the explosion affects, this should be set to "Players".*  
    public ParticleSystem m\_ExplosionParticles;         *// Reference to the particles that will play on explosion.*  
    public float m\_MaxLifeTime = 10f;                    *// The time in seconds before the shell is removed.*  
    public float ShellShotPower;  
    public AudioClip shellBoom;  
    public AudioSource BoomSource;  
  
    *// Use this for initialization*  
    void Start () {  
          
    }  
  
    void OnTriggerEnter (Collider coll) {  
          
        string whoFiredMe;  
  
        *//GameObject pBoomBoom = GameObject.Find ("Fx\_OilSplashHIGH\_Root");*  
        *//pBoomBoom.Play();*  
        *//ParticleSystem pBang = Instantiate (pBoomBoom, gameObject.transform.position, gameObject.transform.rotation ) as ParticleSystem;*  
        *//pBang.Play();*  
*//        Fx\_OilSplashHIGH\_Root*  
  
        ParticleSystem pBoom = Instantiate (m\_ExplosionParticles, gameObject.transform.position, gameObject.transform.rotation ) as ParticleSystem;  
                *// Unparent the particles from the shell.*  
                *//m\_ExplosionParticles.transform.parent = null;*  
        *//pBoom.transform.localScale = new Vector4(1, pBoom.transform.localScale.x,pBoom.transform.localScale.y,pBoom.transform.localScale.z);*  
        pBoom.Play();  
        pBoom.transform.parent = null;  
        BoomSource.Play ();  
        *//Destroy (pBoom, 3f);*  
  
        *//whoever shot can shoot again.*  
        whoFiredMe = gameObject.name;  
        whoFiredMe = whoFiredMe.Replace ("Bullet", "");  
  
        GameObject b = GameObject.Find (whoFiredMe);  
        GameObject g = GameObject.Find ("GameManager");  
  
  
        *// This is the modified part:*  
        float  radius = 5.0f;  
        float power = 10.0f;  
        float totaldamage = 0f;  
        Vector3 explosionPos = transform.position;  
        Collider[] colliders = Physics.OverlapSphere (explosionPos, radius);  
            foreach (Collider hit in colliders) {  
                if (!hit) continue; *// avoid null references (should not occur, but...)*  
            if (hit.tag == "Tank"){  
                *//print("i hit a tank and my strength is" & shotPower*  
*//                print("I hit a tank");*  
                *//print (hit.name);*  
*//                Destroy (hit.transform.parent );*  
                float distance = Vector3.Distance(hit.transform.position, transform.position);  
                float damage = 3.3f / distance ;   
                damage = damage \* ShellShotPower;  
*//                print ("Distance " + distance.ToString() + "; Shot Power " + ShellShotPower.ToString() + "; damage" + damage.ToString());*  
*//                print (ShellShotPower);*  
*//                print (damage);*  
                if (hit.name.StartsWith ("GreyTank")){  
                    GameObject V = hit.gameObject;  
                    V.GetComponent<GreyTankMovement> ().Health = V.GetComponent<GreyTankMovement> ().Health - damage;  
                    totaldamage = totaldamage + damage;  
                }  
                if (hit.name.StartsWith ("Red")){  
                GameObject V = hit.gameObject;  
                V.GetComponent<Damage> ().TakeDamage (damage);  
                    totaldamage = totaldamage + damage;  
                }  
                if (hit.name.StartsWith ("Blue")){  
                    GameObject V = hit.gameObject;  
                    V.GetComponent<Damage> ().TakeDamage (damage);  
                    totaldamage = totaldamage + damage;  
                }  
                if (hit.name.StartsWith ("Green")){  
                    GameObject V = hit.gameObject;  
                    V.GetComponent<Damage> ().TakeDamage (damage);  
                    totaldamage = totaldamage + damage;  
                }  
                if (hit.name.StartsWith ("Yellow")){  
                    GameObject V = hit.gameObject;  
                    V.GetComponent<Damage> ().TakeDamage (damage);  
                    totaldamage = totaldamage + damage;  
                }  
  
*//                hit.GetComponent<Rigidbody>.addexplosionforce(power, explosionPos, radius, 3.0);*  
            }  
        }  
  
        if (whoFiredMe.StartsWith ("Blue") || whoFiredMe.StartsWith ("Red") || whoFiredMe.StartsWith ("Green") || whoFiredMe.StartsWith ("Yellow") ) {  
            *//print (whoFiredMe);*  
            b.GetComponentInChildren<Damage> ().DoDamage ((int)totaldamage);  
        }  
  
        gameObject.GetComponent<Rigidbody> ().AddExplosionForce (1000.0f, transform.position,10.0f,2.0f );  
  
  
  
        *// find the colliders inside a sphere of radius farAreaEffect*  
*//        var colls[] = Physics.OverlapSphere(transform.position, 100);*  
*//        for (var col: Collider in colls){*  
*//            if (col.tag == "British"){ // if it's a bloody British...*  
        *// calculate the distance from the impact...*  
*//                var distance = Vector3.Distance(col.transform.position, transform.position);*  
*//                var damage = farDamage; // assume farDamage initially...*  
*//                if (distance <= closeAreaEffect){*  
*//                    damage = closeDamage; // but if inside close area, change to max damage*  
*//                else*  
*//                    if (distance <= mediumAreaEffect){*  
*//                        damage = mediumDamage; // else if inside medium area, change to medium damage*  
*//                    }*  
        *// apply the selected damage*  
*//                col.SendMessage("ApplyDamage", damage, SendMessageOptions.DontRequireReceiver);*  
*//            }*  
*//        }*  
  
  
        if (coll.tag == "Tank") {  
*//            print ("Tank " + coll.name + " hit by " + gameObject.name);*  
            GameObject t = GameObject.Find (coll.name);  
  
*//    Not using this any more - now look at the update for the tank, and destroy it if the health is less than 0*  
*//    Destroy (t);*  
  
            if (whoFiredMe.StartsWith ("Red")){  
                g.GetComponent<GameManager> ().RedScore = g.GetComponent<GameManager> ().RedScore+10;  
            }  
            if (whoFiredMe.StartsWith ("Blue")){  
                g.GetComponent<GameManager>().BluScore = g.GetComponent<GameManager>().BluScore+10;  
            }  
            if (whoFiredMe.StartsWith ("Green")){  
                g.GetComponent<GameManager>().GrnScore = g.GetComponent<GameManager>().GrnScore +10;  
            }  
            if (whoFiredMe.StartsWith ("Yellow")){  
                g.GetComponent<GameManager>().YelScore = g.GetComponent<GameManager>().YelScore+10;  
            }  
        }  
  
        if (coll.tag == "PlayerTank") {  
            if (whoFiredMe.StartsWith ("Grey")) {  
                if (coll.name.StartsWith("Red") ){  
                    g.GetComponent<GameManager> ().RedScore = g.GetComponent<GameManager> ().RedScore-5;  
                }  
                else  
                {  
                    g.GetComponent<GameManager> ().BluScore = g.GetComponent<GameManager>().BluScore-5;  
                }  
            }  
            else {  
            }  
            if (whoFiredMe.StartsWith("Red") && coll.name.StartsWith("Blue")) {  
                g.GetComponent<GameManager> ().RedScore = g.GetComponent<GameManager> ().RedScore+10;  
                g.GetComponent<GameManager> ().BluScore = g.GetComponent<GameManager>().BluScore-10;  
            }  
            if (whoFiredMe.StartsWith("Blue") && coll.name.StartsWith("Red")) {  
                g.GetComponent<GameManager> ().RedScore = g.GetComponent<GameManager> ().RedScore-10;  
                g.GetComponent<GameManager> ().BluScore = g.GetComponent<GameManager>().BluScore+10;  
                }  
  
        }  
  
        if (b != null) {  
            if (whoFiredMe.StartsWith("Grey")) {  
                    b.GetComponent<GreyTankMovement>().bLiveBullet = false;  
                }  
            if (whoFiredMe.StartsWith ("Red")){  
                    b.GetComponent<RedBullet\_fire>().bLiveBullet = false;  
                }  
            if (whoFiredMe.StartsWith ("Green"))  
                {  
                    b.GetComponent<GreenBullet\_fire>().bLiveBullet = false;  
                }  
            if (whoFiredMe.StartsWith ("Blue"))  
                {  
                    b.GetComponent<BlueBullet\_fire>().bLiveBullet = false;  
                }  
            if (whoFiredMe.StartsWith ("Yellow"))  
                {  
                    b.GetComponent<YellowBullet\_fire>().bLiveBullet = false;  
                }  
                  
                    }  
              
        Destroy(gameObject);  
    }  
  
    void OnCollisionEnter (Collision hit) {  
        print ("Collsision boom");  
    }  
  
  
    *// Update is called once per frame*  
    void Update () {  
          
    }  
}

***Damage***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
  
public class Damage : MonoBehaviour {  
    public bool bCanMove = true;  
    public float Score=0;  
    public float MaxHealth=1000;  
    public float Health=1000;  
    public bool bCanScore = true;  
    public int intRank = 1;  
  
    void Update(){  
        if (Health <= 0 && bCanMove) {  
*//            Destroy (gameObject);*  
            bCanMove = false;  
            bCanScore = false;  
            print ("----------- calculating for" + gameObject.name + " --------------");  
            intRank = intRank + GameObject.Find ("GameManager").GetComponentInChildren<GameManager> ().TanksLeft ();  
            print (gameObject.name +" ranks " + intRank);  
        }  
    }  
  
    public void DoDamage (float DamageGiven){  
        Score = Score + DamageGiven;  
        gameObject.GetComponentInChildren<Text> ().text = Score.ToString();  
    }  
  
    public void TakeDamage (float DamageTaken){  
        Health = Health - DamageTaken;  
        if (Health >= 0) {  
            gameObject.GetComponentInChildren<Slider>().value = Health/MaxHealth;  
        }  
  
    }  
          
}

***GameManager***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
using System.IO;  
  
public class GameManager : MonoBehaviour {  
    public GameObject GreyTank;  
    public GameObject BigGreyTank;  
    public GameObject LittleGreyTank;  
    public GameObject BlueTankPrefab;  
    public GameObject RedTankPrefab;  
    public GameObject GreenTankPrefab;  
    public GameObject YellowTankPrefab;  
    public GameObject Flag;  
    public int numberOfTanks = 20;  
    public int numberOfActiveTanks = 19;  
    public float TimeLimit;  
    public int PlayerCount = 1;  
    public string GameMode = "LastTankStanding";*// could also be "Survival"*  
    private bool bGameIsLive = false;  
    public Dropdown dropdownGameTime;  
    public Dropdown dropdownPlayerCount;  
    public Dropdown dropdownAICount;  
    public Dropdown dropdownGameMode;  
    bool bAllPlayerDead =  false;  
    bool bGameRunning = false;  
    private string[,] ScoreArray = new string[4,2];*//creates an array of 0 to 3 by 0 to 1. Element 0 is Scores, Element 1 is Player Colour.*  
    private string strFileDataPath;  
    private StreamReader srReader;  
    private StreamWriter swWriter;  
    private string[] HighPlayers = new string[10]; *//sets up an array. The array is in fact numbered from 0 to 9.*  
    private string[] HighValues = new string[10];  
    private GameObject BlueTank;  
    private GameObject RedTank;  
    private GameObject GreenTank;  
    private GameObject YellowTank;  
    public int BluScore = 0;  
    public int RedScore = 0;  
    public int GrnScore = 0;  
    public int YelScore = 0;  
    public Text m\_MessageText;  
    public Camera SplashCam;  
    public Camera MainCam;  
    private Camera BluCam;  
    private Camera RedCam;  
    private Camera GrnCam;  
    private Camera YelCam;  
  
  
    *// Use this for initialization*  
    public void Start(){  
        *//Load up the high scores*  
        LoadHighScores();  
        StartCoroutine (Waiting());  
        print ("called init");  
        *//strText = File.ReadAllText();*  
        *//print (strText);*  
        *//HighPlayers[] = strText.ToCharArray();*  
        SplashCam.enabled = true;  
        MainCam.enabled = false;  
        ShowMainMenu ();  
        *//BluCam.enabled = false;*  
        *//RedCam.enabled = false;*  
        *//GrnCam.enabled = false;*  
        *//YelCam.enabled = false;*  
        *//GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;*  
        *//GameObject.Find ("Red\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;*  
        *//GameObject.Find ("Green\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;*  
        *//GameObject.Find ("Yellow\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;*  
  
        Time.timeScale = 0; *//pause the game*  
  
    }  
  
    IEnumerator Waiting(){  
            yield return new WaitForSeconds (5);      
    }  
  
  
  
    public void LoadSettings()  
    {  
        TimeLimit = float.Parse (dropdownGameTime.GetComponentInChildren<Text> ().text) \* 60; *//change the number of minutes to a timelimit, in seconds.*  
*//        print ("-----------------");*  
*//        print (TimeLimit.ToString ());*  
        PlayerCount= int.Parse(dropdownPlayerCount.GetComponentInChildren<Text> ().text);  
        numberOfTanks = int.Parse(dropdownAICount.GetComponentInChildren<Text> ().text);  
        if (dropdownGameMode.value == 1) {  
            GameMode = "Last Tank Standing";  
        }  
        else {  
            GameMode = "Survival";  
        }  
*//        print (numberOfTanks);*  
*//        print (TimeLimit);*  
*//        print (PlayerCount);*  
*//        print (GameMode);*  
  
        Time.timeScale = 1;  
        StartUp ();  
    }  
  
  
    void LoadPlayerTanks (string  SpawnPointName, string TankName, ref GameObject TankPrefab, ref GameObject TankToInstantiate, ref Camera CameraToSet){  
        GameObject tempSpawnPoint;  
  
        tempSpawnPoint= GameObject.Find (SpawnPointName);  
        TankToInstantiate = Instantiate (TankPrefab , tempSpawnPoint.transform.position, tempSpawnPoint.transform.rotation);  
        TankToInstantiate.name = TankName;  
        GameObject.Find (TankName).GetComponentInChildren<Canvas> ().enabled = true;  
        CameraToSet = GameObject.Find (TankName ).GetComponentInChildren<Camera> ();  
        CameraToSet.enabled = true;  
        TankToInstantiate.GetComponentInChildren<Canvas> ().enabled = true;  
    }  
  
  
    public void StartUp () {  
        SplashCam.enabled = false;  
        print ("Startup actioned");  
        MainCam.enabled = true;  
        GameObject Tank;  
        GameObject SpawnPoint;  
  
        *//Destroy any surviving tanks from a previous game. DestroyImmediate is used instead of Destroy, because Destroy waits until the end of the frame.*  
        *//This resulted in the Instantiate method failing unexpectedly - it created a new object, but kept the Camera enabled settings from previously.*  
        DestroyImmediate (GameObject.Find ("Blue\_Tank\_Old"));  
        DestroyImmediate (GameObject.Find ("Red\_Tank\_Old"));  
        DestroyImmediate(GameObject.Find ("Green\_Tank\_Old"));  
        DestroyImmediate(GameObject.Find ("Yellow\_Tank\_Old"));  
  
  
        Tank = GameObject.Find("Blue\_Tank\_Old");  
        if (Tank !=null){  
            *//print(Tank.name);*  
        }  
          
        */\**  
*SpawnPoint = GameObject.Find ("BlueSpawnPoint");*  
*BlueTank = Instantiate (BlueTankPrefab, SpawnPoint.transform.position, SpawnPoint.transform.rotation);*  
*BlueTank.name = "Blue\_Tank\_Old";*  
*GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = true;*  
*BluCam = GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Camera> ();*  
*\*/*  
        *//LoadPlayerTanks ("BlueSpawnPoint", "Blue\_Tank\_Old", ref BlueTankPrefab, ref BlueTank, ref BluCam);*  
        *//LoadPlayerTanks ("RedSpawnPoint", "Red\_Tank\_Old", ref RedTankPrefab, ref RedTank, ref RedCam);*  
        *//LoadPlayerTanks ("GreenSpawnPoint", "Green\_Tank\_Old", ref GreenTankPrefab, ref GreenTank, ref GrnCam);*  
        *//LoadPlayerTanks ("YellowSpawnPoint", "Yellow\_Tank\_Old", ref YellowTankPrefab, ref YellowTank, ref YelCam);*  
  
        switch(PlayerCount){  
        case 1:  
            LoadPlayerTanks ("BlueSpawnPoint", "Blue\_Tank\_Old", ref BlueTankPrefab, ref BlueTank, ref BluCam);  
            BluCam.rect = new Rect (0, 0, 1, 1);  
            *//print ("Blue Cam enabled status is " + BluCam.enabled);*  
            *//GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = true;*  
            break;  
        case 2:  
            LoadPlayerTanks ("BlueSpawnPoint", "Blue\_Tank\_Old", ref BlueTankPrefab, ref BlueTank, ref BluCam);  
            LoadPlayerTanks ("RedSpawnPoint", "Red\_Tank\_Old", ref RedTankPrefab, ref RedTank, ref RedCam);  
            BluCam.rect = new Rect (0, 0, 1, 0.5f);  
            RedCam.rect = new Rect (0f,0.5f,1f,0.5f);  
            break;  
        case 3:  
            LoadPlayerTanks ("BlueSpawnPoint", "Blue\_Tank\_Old", ref BlueTankPrefab, ref BlueTank, ref BluCam);  
            LoadPlayerTanks ("RedSpawnPoint", "Red\_Tank\_Old", ref RedTankPrefab, ref RedTank, ref RedCam);  
            LoadPlayerTanks ("GreenSpawnPoint", "Green\_Tank\_Old", ref GreenTankPrefab, ref GreenTank, ref GrnCam);  
            BluCam.rect = new Rect (0, 0, 1, 0.5f);  
            RedCam.rect = new Rect (0f, 0.5f, 0.5f, 0.5f);  
            GrnCam.rect = new Rect (0.5f, 0.5f, 0.5f, 0.5f);  
            break;  
        case 4:  
            LoadPlayerTanks ("BlueSpawnPoint", "Blue\_Tank\_Old", ref BlueTankPrefab, ref BlueTank, ref BluCam);  
            LoadPlayerTanks ("RedSpawnPoint", "Red\_Tank\_Old", ref RedTankPrefab, ref RedTank, ref RedCam);  
            LoadPlayerTanks ("GreenSpawnPoint", "Green\_Tank\_Old", ref GreenTankPrefab, ref GreenTank, ref GrnCam);  
            LoadPlayerTanks ("YellowSpawnPoint", "Yellow\_Tank\_Old", ref YellowTankPrefab, ref YellowTank, ref YelCam);  
            BluCam.rect = new Rect (0, 0, 0.5f, 0.5f);  
            RedCam.rect = new Rect (0f, 0.5f, 0.5f, 0.5f);  
            GrnCam.rect = new Rect (0.5f, 0.5f, 0.5f, 0.5f);  
            YelCam.rect = new Rect (0.5f, 0, 0.5f, 0.5f);  
*/\*            BluCam.enabled = true;*  
*RedCam.enabled = true;*  
*GrnCam.enabled = true;*  
*YelCam.enabled = true;*  
*BluCam.rect = new Rect (0, 0, 0.5f, 0.5f);*  
*RedCam.rect = new Rect (0f, 0.5f, 0.5f, 0.5f);*  
*GrnCam.rect = new Rect (0.5f, 0.5f, 0.5f, 0.5f);*  
*YelCam.rect = new Rect (0.5f, 0, 0.5f, 0.5f);*  
*GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = true;*  
*GameObject.Find ("Red\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = true;*  
*GameObject.Find ("Green\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = true;*  
*GameObject.Find ("Yellow\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = true;*  
*\*/*  
            break;  
        }  
  
        MainCam.enabled = false;  
              
        Time.timeScale = 1; *//start the game*  
        bGameIsLive = true;  
  
        for (int i = 1; i<=numberOfTanks; i++)  
        {  
            GameObject new\_tank = GameObject.Find ("GreyTank" + i);  
            if (new\_tank == null) {  
                GameObject  spawnPoint = GameObject.Find ("SpawnPoint\_Grey" + i) as GameObject;  
  
                float f = Random.value;  
                if (f < 0.25f) {  
                    GameObject Tanker = Instantiate (BigGreyTank, spawnPoint.transform.position, spawnPoint.transform.rotation) as GameObject;  
                    Tanker.name = "GreyTank" + i;  
                    Tanker.GetComponent<GreyTankMovement> ().Health = 2000;  
                    Tanker.GetComponent<GreyTankMovement> ().ShotPower = 300;  
                    Tanker.GetComponent<GreyTankMovement> ().Speed = 10;  
                } else if (f > 0.62) {  
                    GameObject Tanker = Instantiate (GreyTank, spawnPoint.transform.position, spawnPoint.transform.rotation) as GameObject;  
                    Tanker.name = "GreyTank" + i;  
                    Tanker.GetComponent<GreyTankMovement> ().Health = 500;  
                    Tanker.GetComponent<GreyTankMovement> ().ShotPower = 100;  
                    Tanker.GetComponent<GreyTankMovement> ().Speed = 20;  
                } else {  
                    GameObject Tanker = Instantiate (LittleGreyTank, spawnPoint.transform.position, spawnPoint.transform.rotation) as GameObject;  
                    Tanker.name = "GreyTank" + i;  
                    Tanker.GetComponent<GreyTankMovement> ().Health = 75;  
                    Tanker.GetComponent<GreyTankMovement> ().ShotPower = 50;  
                    Tanker.GetComponent<GreyTankMovement> ().Speed = 30;  
                }  
            }  
        }  
    }  
          
  
    *// Update is called once per frame*  
    void Update () {  
        string OutPutScore;  
        int intPlayersAlive;  
        int intEnemiesAlive;  
  
*//        GameObject s;*  
        float f;  
        if (GameMode == "Survival"){  
            for (int i = 1; i <= numberOfTanks; i++) {  
                GameObject new\_tank = GameObject.Find ("GreyTank" + i);  
                if (new\_tank == null) {  
                    GameObject spawnPoint = GameObject.Find ("SpawnPoint\_Grey" + i) as GameObject;  
                    GameObject Tank = Instantiate (GreyTank, spawnPoint.transform.position, spawnPoint.transform.rotation) as GameObject;  
                    Tank.GetComponent<GreyTankMovement> ().Health = 100;  
                    Tank.GetComponent<GreyTankMovement> ().ShotPower = 75;  
                    Tank.GetComponent<GreyTankMovement> ().Speed = 20;  
                    Tank.name = "GreyTank" + i;  
                }  
            }  
        }  
  
        if (bGameIsLive) {  *//work out whether we should end the game. end if (1) timer runs out; (2) no players left, (3) only one player and no enemy tanks left.*  
            TimeLimit -= Time.deltaTime;  
            *//print (TimeLimit.ToString ());*  
            GameObject s = GameObject.FindGameObjectWithTag ("Timer");  
            s.GetComponentInChildren<Text> ().text = TimeLimit.ToString ();  
            if (TimeLimit < 0) {  
                GameOver ();  
            }  
                  
  
            intPlayersAlive = 0;  
            intEnemiesAlive = 0;  
            GameObject[] w = GameObject.FindGameObjectsWithTag ("Tank");  
            foreach (GameObject y in w) {  
*//                print (y.name);*  
                if (y.name.EndsWith ("\_Tank\_Old")) {  
                    if (y.GetComponent<Damage> ().bCanMove == true) {  
                        *//print ("Found alive one");*  
                        intPlayersAlive++;  
                    }  
                }  
                if (y.name.StartsWith ("Grey")) {  
                    intEnemiesAlive++;  
                }  
            }  
  
            *//print ("Players alive: " + intPlayersAlive.ToString() + ", enemies alive: " + intEnemiesAlive.ToString());*  
            if (intPlayersAlive==0){  
                GameOver();  
            }  
            if (intPlayersAlive == 1 && intEnemiesAlive == 0) {  
                GameOver ();  
            }  
  
            GameObject[] p = GameObject.FindGameObjectsWithTag ("boom");  
            foreach (GameObject go in p) {  
                if (go.transform.parent == null) {  
                    Destroy (go, 3);  
                }  
            }  
        }  
    }  
  
    public void ShowMainMenu(){  
        bGameIsLive = false;  
        print ("showing main menu ");  
        ChangeCanvasDisplay ("GameOverTag", 0);  
        ChangeCanvasDisplay ("HighScoreTag", 0);  
        ChangeCanvasDisplay ("MainScreenTag", 1);  
        ChangeCanvasDisplay ("NewHighScoreTag", 0);  
    }  
  
    public void ShowNewHighScore(){  
        bGameIsLive = false;  
        print ("showing new high scores");  
        ChangeCanvasDisplay ("GameOverTag", 0);  
        ChangeCanvasDisplay ("HighScoreTag", 0);  
        ChangeCanvasDisplay ("MainScreenTag", 0);  
        ChangeCanvasDisplay ("NewHighScoreTag", 1);  
    }  
  
    public void ShowHighScores(){  
        bGameIsLive = false;  
        print ("showing high scores");  
        ChangeCanvasDisplay ("GameOverTag", 0);  
        ChangeCanvasDisplay ("HighScoreTag", 1);  
        ChangeCanvasDisplay ("MainScreenTag", 0);  
        ChangeCanvasDisplay ("NewHighScoreTag", 0);  
  
        for (int i = 0; i <= 9; i++) {  
            GameObject.Find ("PlayerName" + i.ToString()).GetComponent<Text> ().text = HighPlayers[i];  
            GameObject.Find ("PlayerScore"+i.ToString()).GetComponent<Text> ().text = HighValues[i];  
        }  
    }  
  
    public void ShowGameOver(){  
        string strTemp;  
        bGameIsLive = false;  
        print ("showing game over");  
        ChangeCanvasDisplay ("GameOverTag", 1);  
        ChangeCanvasDisplay ("HighScoreTag", 0);  
        ChangeCanvasDisplay ("MainScreenTag", 0);  
        ChangeCanvasDisplay ("NewHighScoreTag", 0);  
  
        *//ScoreArray is now sorted from 0 (Smalles) to 3 (Biggest)*  
        *//Clear out the old game results*  
        GameObject.Find ("gPlayerName1").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerScore1").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerName2").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerScore2").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerName3").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerScore3").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerName4").GetComponent<Text> ().text = "";  
        GameObject.Find ("gPlayerScore4").GetComponent<Text> ().text = "";  
  
  
        *//now put in the game scores for the number of players that were active*  
        for (int i = 0; i <PlayerCount ; i++) {  
            *//print (GameObject.Find ("gPlayerName" + (i + 1)).name);*  
            *//print (GameObject.Find ("gPlayerScore" + (i + 1)).name);*  
            GameObject.Find ("gPlayerName" + (i + 1)).GetComponent<Text> ().text = ScoreArray [i, 0];  
            GameObject.Find ("gPlayerScore" + (i + 1)).GetComponent<Text> ().text = ScoreArray [i, 1];  
        }  
           
  
*/\*        switch (PlayerCount){*  
*case 1:*  
*case 3:*  
*case 4:*  
*GameObject.Find ("gPlayerName1").GetComponent<Text> ().text = "Blue Player";*  
*GameObject.Find ("gPlayerScore1").GetComponent<Text> ().text = GameObject.Find ("Blue\_Tank\_Old").GetComponent<Damage> ().Score.ToString();*  
*GameObject.Find ("gPlayerName2").GetComponent<Text> ().text = "Red Player";*  
*GameObject.Find ("gPlayerScore2").GetComponent<Text> ().text = GameObject.Find ("Red\_Tank\_Old").GetComponent<Damage> ().Score.ToString();*  
*GameObject.Find ("gPlayerName3").GetComponent<Text> ().text = "Green Player";*  
*GameObject.Find ("gPlayerScore3").GetComponent<Text> ().text = GameObject.Find ("Green\_Tank\_Old").GetComponent<Damage> ().Score.ToString();*  
*GameObject.Find ("gPlayerName4").GetComponent<Text> ().text = "Yellow Player";*  
*GameObject.Find ("gPlayerScore4").GetComponent<Text> ().text = GameObject.Find ("Yellow\_Tank\_Old").GetComponent<Damage> ().Score.ToString();*  
*break;*  
*}*  
*\*/*  
    }  
  
    void GameEnding()  
    {  
        *//load up the player scores into an array of scores for sorting. A switch statement doesn't work because C# switch doesn't allow fall-through.*  
        *//this is quite messy, I need to think of a way to clean it up.*  
        *//part of the problem is that in Survival, high scores are good, and in Last Tank Standing low scores are good.*  
  
        if (GameMode == "Survival") {  
            if (PlayerCount == 4) {  
                ScoreArray [3, 0] = GameObject.Find ("Yellow\_Tank\_Old").GetComponent<Damage> ().Score.ToString ();  
                ScoreArray [3, 0] = GameObject.Find ("Yellow\_Tank\_Old").GetComponent<Damage> ().intRank.ToString ();  
            }  
            if (PlayerCount >= 3) {  
                ScoreArray [2, 0] = GameObject.Find ("Green\_Tank\_Old").GetComponent<Damage> ().Score.ToString ();  
            }  
            if (PlayerCount >= 2) {  
                ScoreArray [1, 0] = GameObject.Find ("Red\_Tank\_Old").GetComponent<Damage> ().Score.ToString ();  
            }  
            ScoreArray [0, 0] = GameObject.Find ("Blue\_Tank\_Old").GetComponent<Damage> ().Score.ToString ();  
        } else {  
            if (PlayerCount == 4) {  
                ScoreArray [3, 0] = GameObject.Find ("Yellow\_Tank\_Old").GetComponent<Damage> ().intRank.ToString ();  
            }  
            if (PlayerCount >= 3) {  
                ScoreArray [2, 0] = GameObject.Find ("Green\_Tank\_Old").GetComponent<Damage> ().intRank.ToString ();  
            }  
            if (PlayerCount >= 2) {  
                ScoreArray [1, 0] = GameObject.Find ("Red\_Tank\_Old").GetComponent<Damage> ().intRank.ToString ();  
            }  
            ScoreArray [0, 0] = GameObject.Find ("Blue\_Tank\_Old").GetComponent<Damage> ().intRank.ToString ();  
        }  
  
  
*/\*        for (int i = 0; i <= 3; i++) {*  
*if (ScoreArray [i, 0] == null) {*  
*switch (GameMode ) {*  
*case "Survival":*  
*ScoreArray [i, 0] = "0";*  
*break;*  
*default:*  
*ScoreArray [i, 0] = "25";*  
*break;*  
*}*  
*}*  
*}\*/*  
  
        ScoreArray [0, 1] = "Blue Player";  
        ScoreArray [1, 1] = "Red Player";  
        ScoreArray [2, 1] = "Green Player";  
        ScoreArray [3, 1] = "Yellow Player";  
  
        *//print ("got this far");*  
        *//print ("check");*  
  
        if (GameMode == "Survival") {  
            *//print ("check");*  
            BubbleSortHighToLow (ScoreArray);  
            print (ScoreArray [0, 0]);  
            print (HighValues [9]);  
            *//print ("check");*  
            if (int.Parse(ScoreArray[0,0]) > int.Parse(HighValues[9])) {  
                *//print ("check");*  
                ShowNewHighScore(); *//open a dialog for the best player to enter their score.*  
                *//ShowGameOver();*  
            }  
            else {  
                *//print ("check");*  
                ShowGameOver();  
            }  
        }  
        else  
        {  
            *//print ("check");*  
            BubbleSortLowToHigh (ScoreArray);  
            *//print ("check");*  
            ShowGameOver();          
        }  
    }  
  
    void LoadHighScores(){  
        *//load up the high scores table. This will give 2 arrays - HighPlayers(10) and HighValues(10).*  
        *//string[] strTemp;*  
        strFileDataPath = Path.GetDirectoryName (Application.dataPath) + "/HighScores.txt";  
        *//print (strFileDataPath);*  
        string[] strText = File.ReadAllLines  (strFileDataPath);  
        for (int i = 0; i <= 9; i++) {  
            *//print (strText [i]);*  
            string [] strTemp = strText[i].Split(',');  
            *//print (strTemp[0]);*  
            *//print (strTemp[1]);*  
            *//print (HighValues.GetLowerBound(0));*  
            HighValues[i] = strTemp[0];  
            HighPlayers[i] = strTemp[1];  
            *//print (HighPlayers[i]);*  
            *//print (HighValues[i]);*  
            *//print ("----------");*  
        }  
    }  
  
    public void UpdateHighScores(){  
        int tmpScore;  
        string tmpName;  
        tmpScore = int.Parse (ScoreArray [0, 0]);  
        tmpName = GameObject.Find ("NewHighScoreInputText").GetComponentInChildren<Text>().text;  
  
        for (int i = 0; i <= 9; i++) {  
            if(tmpScore>int.Parse(HighValues[i])){  
                for (int j = 8 ; j >=i; j--) {  
                    HighValues [j + 1] = HighValues [j];  
                    HighPlayers [j + 1] = HighPlayers [j];  
                }  
                HighValues [i] = tmpScore.ToString ();  
                HighPlayers [i] = tmpName;  
                break;  
            }  
        }  
        *//write back to the high score table*  
        SaveHighScores();  
        ShowHighScores();  
    }  
  
    void SaveHighScores(){  
        *//load up the high scores table. This will give 2 arrays - HighPlayers(10) and HighValues(10).*  
        string strTemp;  
        strTemp = "";  
        strFileDataPath = Path.GetDirectoryName (Application.dataPath) + "/HighScores.txt";  
  
        *//print (strFileDataPath);*  
        string[] strText = File.ReadAllLines (strFileDataPath);  
  
        for (int i = 0; i <= 9; i++) {  
            strTemp = strTemp + HighValues [i] + "," + HighPlayers [i] + "\r\n"; *// \r\n means carriage return and new line*  
        }  
        File.WriteAllText (strFileDataPath, strTemp);  
    }  
  
    void ChangeCanvasDisplay(string strChangeTag, int intValue){  
        GameObject[] xx; *//an array of gameobjects that we'll put the game object's with the relevant tag into.*  
        xx = GameObject.FindGameObjectsWithTag(strChangeTag);  
        foreach(GameObject yy in xx){  
            yy.transform.localScale = new Vector3 (intValue, intValue, intValue);  
        }  
    }  
          
    void GameOver(){  
        *//print (TimeLimit.ToString ());*  
        Time.timeScale = 0;  
        SplashCam.enabled = true;  
        MainCam.enabled = false;  
        switch (PlayerCount)  
        {  
        case 1:  
            BluCam.enabled = false;  
            GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            break;  
        case 2:  
            BluCam.enabled = false;  
            RedCam.enabled = false;  
            GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            GameObject.Find ("Red\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            break;  
        case 3:  
            BluCam.enabled = false;  
            RedCam.enabled = false;  
            GrnCam.enabled = false;  
            GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            GameObject.Find ("Red\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            GameObject.Find ("Green\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            break;  
        case 4:  
            BluCam.enabled = false;  
            RedCam.enabled = false;  
            GrnCam.enabled = false;  
            YelCam.enabled = false;  
            GameObject.Find ("Blue\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            GameObject.Find ("Red\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            GameObject.Find ("Green\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            GameObject.Find ("Yellow\_Tank\_Old").GetComponentInChildren<Canvas> ().enabled = false;  
            break;  
        }  
        GameEnding ();  
  
    }  
  
    void BubbleSortHighToLow( string[,] ScoreArray)  
    {  
        string strTemp0;  
        string strTemp1;  
        bool bSwap;  
        do { *//bubble sort biggest to smallest*  
            bSwap = false;  
            for (int i=0; i<(PlayerCount-1);i++)  
            {  
                if(int.Parse (ScoreArray[i,0])<int.Parse(ScoreArray[i+1,0])){  
                    strTemp0 = ScoreArray[i,0];  
                    strTemp1 = ScoreArray[i,1];  
                    ScoreArray[i,0] = ScoreArray[i+1,0];  
                    ScoreArray[i,1] = ScoreArray[i+1,1];  
                    ScoreArray[i+1,0]= strTemp0;  
                    ScoreArray[i+1,1]=strTemp1;  
                    bSwap=true;  
                }  
            }  
        } while (bSwap==true);  
    }  
  
    void BubbleSortLowToHigh(string[,] ScoreArray)  
    {  
        string strTemp0;  
        string strTemp1;  
        bool bSwap;  
        */\*        print ("BA");*  
*print (ScoreArray [0, 0]);*  
*print (ScoreArray [0, 1]);*  
*print (ScoreArray [1, 0]);*  
*print (ScoreArray [1, 1]);*  
*print (ScoreArray [2, 0]);*  
*print (ScoreArray [2, 1]);*  
*print (ScoreArray [3, 0]);*  
*print (ScoreArray [3, 1]);*  
*\*/*  
        do { *//bubble sort biggest to smallest*  
            bSwap = false;  
            for (int i=0; i<PlayerCount-1;i++)  
            {  
                if(int.Parse (ScoreArray[i,0])>int.Parse(ScoreArray[i+1,0])){  
                    strTemp0 = ScoreArray[i,0];  
                    strTemp1 = ScoreArray[i,1];  
                    ScoreArray[i,0] = ScoreArray[i+1,0];  
                    ScoreArray[i,1] = ScoreArray[i+1,1];  
                    ScoreArray[i+1,0]= strTemp0;  
                    ScoreArray[i+1,1]=strTemp1;  
                    bSwap=true;  
                }  
            }  
        } while (bSwap==true);  
    }  
  
    public void QuitButton(){  
        print ("Quitting");  
        Application.Quit ();  
    }  
  
    public int TanksLeft (){  
        int intTankCount;  
        intTankCount = 0;*//start at 1, because the player tank that just died still counts.*  
        GameObject[] xx; *//an array of gameobjects that we'll put the game object's with the relevant tag into.*  
        xx = GameObject.FindGameObjectsWithTag("Tank");  
        foreach(GameObject yy in xx){  
            if (yy.name.EndsWith ("\_Tank\_Old")) { *//this is a player tank. Check if it can still move (i.e. it's live)*  
                if (yy.GetComponentInChildren<Damage> ().bCanMove == true) {  
                    intTankCount++;  
                    print (yy.name);  
                }  
            }   
            else {*// it's a computer tank. no need to check any further*  
                print(yy.name);  
                intTankCount++;  
            }  
        }  
        return intTankCount;  
    }  
  
}

***GreenBullet\_fire***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class GreenBullet\_fire : MonoBehaviour {  
    public AudioClip shotFire;  
    public AudioSource FiringSource;  
    public bool bLiveBullet = false;  
    public Rigidbody bullet;  
    public Transform bullet\_transform;  
    public float bulletSpeed ;  
  
    *// Use this for initialization*  
    void Start () {  
        FiringSource.clip = shotFire;  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanScore == true) {  
            if (Input.GetKey (KeyCode.Space)) {  
                if (!bLiveBullet) {  
                    Shoot ();  
                }  
            }  
        }  
  
        if (Input.GetKey(KeyCode.I)) {  
            print ("Bullet\_transform position: " + bullet\_transform.position);  
        }  
  
    }

  void Shoot()  
    {  
        Rigidbody newBullet = Instantiate (bullet, bullet\_transform.position, bullet\_transform.rotation) as Rigidbody;  
        newBullet.velocity = bulletSpeed \* bullet\_transform.forward;  
        newBullet.name = gameObject.name + "Bullet";  
        bLiveBullet = true;  
        newBullet.gameObject.GetComponent<BulletCollider> ().ShellShotPower = gameObject.GetComponent<GreenTankMovement> ().ShotPower;  
        FiringSource.Play ();  
    }  
}

***GreenTankMovement***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
  
public class GreenTankMovement : MonoBehaviour   
{  
    public float ShotPower;  
    public float Speed;  
    public int FlagCount = 0;  
    private Rigidbody myRigidbody;  
    public float mySpeed;  
    private float myTurnDirection;  
    public float myTurningSpeed = 105f;  
  
    *// Use this for initialization*  
    void Start () {  
        myRigidbody = GetComponent<Rigidbody> ();  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanMove == true) {          
            if (Input.GetKey (KeyCode.Y)) {  
                *// this is where I move forward*  
                mySpeed = 20.0f;  
                Move (mySpeed);  
            }  
  
            if (Input.GetKey (KeyCode.H)) {  
                mySpeed = -20.0f;  
                Move (mySpeed);  
            }  
            if (Input.GetKey (KeyCode.G)) {  
                *//rotate left*  
                myTurnDirection = -1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            if (Input.GetKey (KeyCode.J)) {  
                *//rotate right*  
                myTurnDirection = 1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
*//        if (Input.GetKey (KeyCode.Space)) {*  
*//            //This will destroy the tank!*  
*//            Destroy (gameObject);*  
*//        }*  
  
            */\*if (Input.GetKey (KeyCode.R)) {*  
*//test raycast*  
*Debug.DrawRay(myRigidbody.position ,myRigidbody.transform.forward \* 100f, Color.red);*  
*print (myRigidbody.name) ;*  
*print (myRigidbody.transform.position);*  
*}\*/*  
  
        }  
    }  
  
    private void Move(float mySpeed)  
    {  
        *// Create a vector in the direction the tank is facing with a magnitude based on the input, speed and the time between frames.*  
        Vector3 movement = transform.forward \*  mySpeed \* Time.deltaTime;  
  
        *// Apply this movement to the rigidbody's position.*  
        myRigidbody.MovePosition(myRigidbody.position + movement);  
    }  
  
  
    void OnCollisionEnter(Collision coll) {  
        *//print ("collided with " + collision.collider.ToString ());*  
        *//print ("collided with " + collision.collider.tag);*  
        GameObject g;  
        *//print (coll.gameObject.name);*  
        if (coll.gameObject.tag == "FlagPole") {  
            FlagCount++;  
            print ("i hit " );  
            Destroy (coll.transform.parent.gameObject );  
            g = GameObject.Find ("GameManager");  
*//            g.GetComponent<GameManager>().BluScore =g.GetComponent<GameManager>().BluScore + 3;*  
            *//Destroy (collision.collider, 0);*  
        }  
    }          
  
  
    private void Turn (float myTurnDirection, float myTurningSpeed)  
    {  
        *// Determine the number of degrees to be turned based on the input, speed and time between frames.*  
        float turn = myTurnDirection \* myTurningSpeed \* Time.deltaTime;  
  
        *// Make this into a rotation in the y axis.*  
        Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);  
  
        *// Apply this rotation to the rigidbody's rotation.*  
        myRigidbody.MoveRotation (myRigidbody.rotation \* turnRotation);  
    }  
  
  
  
  
}

***GreyTankMovement***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class GreyTankMovement : MonoBehaviour {  
    public float Health;  
    public float ShotPower;  
    public float Speed;  
    public bool bLiveBullet = false;  
    public bool bMovingForward = true ;  
    public string myTargetName;  
    public bool bMovingLeft = false ;  
    private Rigidbody myRigidbody;  
    public float mySpeed;  
    private float myTurnDirection;  
    public float myTurningSpeed = 105f;  
    public bool b\_fired = false;  
    public Rigidbody bullet;  
    public Transform bullet\_transform;  
    public float bulletSpeed ;  
  
        *// Use this for initialization*  
        void Start () {  
            int fForwardBack;  
            int fLeftRight;  
            myRigidbody = GetComponent<Rigidbody> ();  
  
            fForwardBack = Random.Range(0,11);  
            fLeftRight= Random.Range(0,11);  
  
            if (fForwardBack>5) {bMovingForward = false;}  
            if (fLeftRight>5) {bMovingLeft = true;}  
        }  
  
  
        *// Update is called once per frame*  
        void Update () {  
        if (Health < 0) {  
            Destroy (gameObject,1);  
        }  
        if (bLiveBullet) {  
            EvasiveMove ();  
        } else {  
            BeAggressive ();  
        }  
  
    }

    private void BeAggressive()  
    {  
  
        RaycastHit hit;  
        *// raycast - can I see a target?*  
        if (Physics.Raycast (myRigidbody.position, myRigidbody.transform.forward, out hit, 200f)) {  
            *//print ("I spotted a " + hit.collider.tag);*  
            if ((hit.collider.tag == "Tank") || (hit.collider.tag == "PlayerTank")) {  
                if (hit.distance < 100f) { *//in range?*  
                    *//in range! Fire!*  
                    *//print ("I want to shoot at " + hit.collider.name);*  
                    Shoot ();  
                } else {  
                    *//Out of range - move toward target.*  
                    myTargetName = hit.collider.name ;  *//use this to hunt afterwards, not raycasting*  
                    Move(Speed);  
                }  
            }  
            else{  
                *//if I can't see a target, move in a big left circle)\*/*  
                Move (Speed);  
                Turn (-1f, myTurningSpeed);  
            }  
        } else {  
            *//if I can't see a target, move in a big left circle)\*/*  
            Move (Speed);  
            Turn (-1f, myTurningSpeed);  
        }  
            
    }  
  
    private void EvasiveMove()  
    {  
        int RandomNumber;  
  
        *//Forward backness*  
        RandomNumber = Random.Range(1,1001);  
        if(RandomNumber>=995)   
        {  
            bMovingForward = !bMovingForward;  
        }  
        *//left rightness*  
        RandomNumber = Random.Range(1,11);  
        if(RandomNumber>=9)   
        {  
            bMovingLeft = !bMovingLeft;  
        }  
  
        if (bMovingForward)   
        {  
            mySpeed = Speed;  
        }  
        else {  
            mySpeed = -Speed;  
        }  
  
        if (bMovingLeft)   
        {  
            myTurnDirection = -1f;  
        }  
        else {  
            myTurnDirection = 1f;  
        }  
  
        Move (mySpeed);  
        Turn (myTurnDirection, myTurningSpeed);  
    }  
  
          
        private void Move(float mySpeed)  
        {  
            *// Create a vector in the direction the tank is facing with a magnitude based on the input, speed and the time between frames.*  
            Vector3 movement = transform.forward \*  mySpeed \* Time.deltaTime;  
  
            *// Apply this movement to the rigidbody's position.*  
            myRigidbody.MovePosition(myRigidbody.position + movement);  
        }  
  
  
        void OnCollisionEnter(Collision collision) {  
            *//print ("collided with " + collision.collider.ToString ());*  
        if (collision.collider.tag == "Wall") {  
            bMovingForward = !bMovingForward;  
        }  
  
  
        }          
  
  
        private void Turn (float myTurnDirection, float myTurningSpeed)  
        {  
            *// Determine the number of degrees to be turned based on the input, speed and time between frames.*  
            float turn = myTurnDirection \* myTurningSpeed \* Time.deltaTime;  
  
            *// Make this into a rotation in the y axis.*  
            Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);  
  
            *// Apply this rotation to the rigidbody's rotation.*  
            myRigidbody.MoveRotation (myRigidbody.rotation \* turnRotation);  
        }  
          
  
    void Shoot()  
    {  
        Rigidbody newBullet = Instantiate (bullet, bullet\_transform.position, bullet\_transform.rotation) as Rigidbody;  
        newBullet.velocity = bulletSpeed \* bullet\_transform.forward;  
        newBullet.name = gameObject.name + "Bullet";  
        bLiveBullet = true;  
        newBullet.gameObject.GetComponent<BulletCollider>().ShellShotPower = ShotPower;  
  
    }  
}

***PlayerSettingsScript***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI; *//this one had to be added in so we can reference the Settings Dropdowns*  
  
public class PlayerSettingsScript : MonoBehaviour {  
    public Dropdown dropdownGameTime;  
    public Dropdown dropdownPlayerCount;  
    public Dropdown dropdownAICount;  
    public Dropdown dropdownGameMode;  
  
    public int numberOfTanks = 20;  
    public int numberOfActiveTanks = 19;  
    public int TimeLimit;  
    public int PlayerCount = 1;  
    public string GameMode = "Last Tank Standing";*// could also be "Survival"*  
  
    public void LoadSettings()  
    {  
        TimeLimit = int.Parse (dropdownGameTime.GetComponentInChildren<Text> ().text);  
        PlayerCount= int.Parse(dropdownPlayerCount.GetComponentInChildren<Text> ().text);  
        numberOfTanks = int.Parse(dropdownAICount.GetComponentInChildren<Text> ().text);  
        if (dropdownGameMode.value == 1) {  
            GameMode = "Last Tank Standing";  
        }  
        else {  
            GameMode = "Survival";  
        }  
        print (numberOfTanks);  
        print (TimeLimit);  
        print (PlayerCount);  
        print (GameMode);  
  
        GameObject g = GameObject.Find ("GameManager");  
  
    }  
  
    *// Use this for initialization*  
    void Start () {  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
          
    }  
}

***RedBullet\_fire***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class RedBullet\_fire : MonoBehaviour {  
    public AudioClip shotFire;  
    public AudioSource FiringSource;  
    public bool bLiveBullet = false;  
    public Rigidbody bullet;  
    public Transform bullet\_transform;  
    public float bulletSpeed ;  
  
    *// Use this for initialization*  
    void Start () {  
        FiringSource.clip = shotFire;  
    }  
      
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanScore == true) {  
            if (Input.GetKey (KeyCode.RightControl)) {  
                if (!bLiveBullet) {  
                    Shoot ();  
                }  
            }  
        }  
  
        if (Input.GetKey(KeyCode.I)) {  
            print ("Bullet\_transform position: " + bullet\_transform.position);  
        }  
  
        }  
  
    void Shoot()  
    {  
        Rigidbody newBullet = Instantiate (bullet, bullet\_transform.position, bullet\_transform.rotation) as Rigidbody;  
        newBullet.velocity = bulletSpeed \* bullet\_transform.forward;  
        newBullet.name = gameObject.name + "Bullet";  
        bLiveBullet = true;  
        newBullet.gameObject.GetComponent<BulletCollider> ().ShellShotPower = gameObject.GetComponent<RedTankMovement> ().ShotPower;  
        FiringSource.Play ();  
    }  
}

***RedTankMovement***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
  
public class RedTankMovement : MonoBehaviour   
    {  
    public float ShotPower;  
    public float Speed;  
    public int FlagCount = 0;  
    private Rigidbody myRigidbody;  
    public float mySpeed;  
    private float myTurnDirection;  
    public float myTurningSpeed = 105f;  
  
    *// Use this for initialization*  
    void Start () {  
        myRigidbody = GetComponent<Rigidbody> ();  
    }  
      
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanMove == true) {  
            if (Input.GetKey (KeyCode.UpArrow)) {  
                *// this is where I move forward*  
                mySpeed = 20.0f;  
                Move (mySpeed);  
            }  
  
            if (Input.GetKey (KeyCode.DownArrow)) {  
                mySpeed = -20.0f;  
                Move (mySpeed);  
            }  
            if (Input.GetKey (KeyCode.LeftArrow)) {  
                *//rotate left*  
                myTurnDirection = -1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            if (Input.GetKey (KeyCode.RightArrow)) {  
                *//rotate right*  
                myTurnDirection = 1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            if (Input.GetKey (KeyCode.RightShift)) {  
                *//This will destroy the tank!*  
                Destroy (gameObject);  
            }  
  
            if (Input.GetKey (KeyCode.R)) {  
                *//test raycast*  
                Debug.DrawRay (myRigidbody.position, myRigidbody.transform.forward \* 100f, Color.red);  
                print (myRigidbody.name);  
                print (myRigidbody.transform.position);  
            }  
  
        }  
    }  
  
    private void Move(float mySpeed)  
    {  
        *// Create a vector in the direction the tank is facing with a magnitude based on the input, speed and the time between frames.*  
        Vector3 movement = transform.forward \*  mySpeed \* Time.deltaTime;  
  
        *// Apply this movement to the rigidbody's position.*  
        myRigidbody.MovePosition(myRigidbody.position + movement);  
    }  
           
  
    void OnCollisionEnter(Collision coll) {  
        GameObject g;  
        *//print ("collided with " + collision.collider.ToString ());*  
        *//print ("collided with " + collision.collider.tag);*  
  
        if (coll.gameObject.tag == "FlagPole") {  
            FlagCount++;  
            print ("i hit " );  
            Destroy (coll.transform.parent.gameObject );  
            g = GameObject.Find ("GameManager");  
            g.GetComponent<GameManager>().RedScore =g.GetComponent<GameManager>().RedScore + 3;   
            *//Destroy (collision.collider, 0);*  
        }  
    }          
  
  
    private void Turn (float myTurnDirection, float myTurningSpeed)  
    {  
        *// Determine the number of degrees to be turned based on the input, speed and time between frames.*  
        float turn = myTurnDirection \* myTurningSpeed \* Time.deltaTime;  
  
        *// Make this into a rotation in the y axis.*  
        Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);  
  
        *// Apply this rotation to the rigidbody's rotation.*  
        myRigidbody.MoveRotation (myRigidbody.rotation \* turnRotation);  
    }  
  
  
  
  
}

***YellowBullet\_fire***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class YellowBullet\_fire : MonoBehaviour {  
    public AudioClip shotFire;  
    public AudioSource FiringSource;  
    public bool bLiveBullet = false;  
    public Rigidbody bullet;  
    public Transform bullet\_transform;  
    public float bulletSpeed ;  
  
    *// Use this for initialization*  
    void Start () {  
        FiringSource.clip = shotFire;  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanScore == true) {  
            if (Input.GetKey (KeyCode.Keypad0)) {  
                if (!bLiveBullet) {  
                    Shoot ();  
                }  
            }  
        }  
  
        if (Input.GetKey(KeyCode.I)) {  
            print ("Bullet\_transform position: " + bullet\_transform.position);  
        }  
  
    }  
  
    void Shoot()  
    {  
        Rigidbody newBullet = Instantiate (bullet, bullet\_transform.position, bullet\_transform.rotation) as Rigidbody;  
        newBullet.velocity = bulletSpeed \* bullet\_transform.forward;  
        newBullet.name = gameObject.name + "Bullet";  
        bLiveBullet = true;  
        newBullet.gameObject.GetComponent<BulletCollider> ().ShellShotPower = gameObject.GetComponent<YellowTankMovement> ().ShotPower;  
        FiringSource.Play ();  
    }  
}

***YellowTankMovement***

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
  
public class YellowTankMovement : MonoBehaviour   
{  
    public float ShotPower;  
    public float Speed;  
    public int FlagCount = 0;  
    private Rigidbody myRigidbody;  
    public float mySpeed;  
    private float myTurnDirection;  
    public float myTurningSpeed = 105f;  
  
    *// Use this for initialization*  
    void Start () {  
        myRigidbody = GetComponent<Rigidbody> ();  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
        if (gameObject.GetComponentInChildren<Damage> ().bCanMove == true) {  
            if (Input.GetKey (KeyCode.Keypad8)) {  
                *// this is where I move forward*  
                mySpeed = 20.0f;  
                Move (mySpeed);  
            }  
  
            if (Input.GetKey (KeyCode.Keypad5)) {  
                mySpeed = -20.0f;  
                Move (mySpeed);  
            }  
            if (Input.GetKey (KeyCode.Keypad4)) {  
                *//rotate left*  
                myTurnDirection = -1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            if (Input.GetKey (KeyCode.Keypad6)) {  
                *//rotate right*  
                myTurnDirection = 1f;  
                Turn (myTurnDirection, myTurningSpeed);  
            }  
  
            *//        if (Input.GetKey (KeyCode.RightShift)) {*  
            *//            //This will destroy the tank!*  
            *//            Destroy (gameObject);*  
            *//        }*  
  
            */\*if (Input.GetKey (KeyCode.R)) {*  
*//test raycast*  
*Debug.DrawRay(myRigidbody.position ,myRigidbody.transform.forward \* 100f, Color.red);*  
*print (myRigidbody.name) ;*  
*print (myRigidbody.transform.position);*  
*}\*/*  
        }  
    }  
  
    private void Move(float mySpeed)  
    {  
        *// Create a vector in the direction the tank is facing with a magnitude based on the input, speed and the time between frames.*  
        Vector3 movement = transform.forward \*  mySpeed \* Time.deltaTime;  
  
        *// Apply this movement to the rigidbody's position.*  
        myRigidbody.MovePosition(myRigidbody.position + movement);  
    }  
  
  
    void OnCollisionEnter(Collision coll) {  
        *//print ("collided with " + collision.collider.ToString ());*  
        *//print ("collided with " + collision.collider.tag);*  
        GameObject g;  
        *//print (coll.gameObject.name);*  
        if (coll.gameObject.tag == "FlagPole") {  
            FlagCount++;  
            print ("i hit " );  
            Destroy (coll.transform.parent.gameObject );  
            g = GameObject.Find ("GameManager");  
*//            g.GetComponent<GameManager>().BluScore =g.GetComponent<GameManager>().BluScore + 3;*  
            *//Destroy (collision.collider, 0);*  
        }  
    }          
  
  
    private void Turn (float myTurnDirection, float myTurningSpeed)  
    {  
        *// Determine the number of degrees to be turned based on the input, speed and time between frames.*  
        float turn = myTurnDirection \* myTurningSpeed \* Time.deltaTime;  
  
        *// Make this into a rotation in the y axis.*  
        Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);  
  
        *// Apply this rotation to the rigidbody's rotation.*  
        myRigidbody.MoveRotation (myRigidbody.rotation \* turnRotation);  
    }  
  
  
  
  
}