# Title Page

Student’s Name: Jonathan G Watson

First Supervisor: Ian

# Project Title: Modding Toolkit For Robot Arena 2

Project Statement:

This project is based on the game Robot Arena 2. The main Objective is to create a tool kit that allows the user of this kit to create their own basic computer-controlled bots to fight in game. This because the game itself is quite hard to edit without high knowledge of how the game works and a simple knowledge of a.i. and python. Therefore, most people who want to create custom bots to fight against in game can’t due to limited knowledge.

The act of changing this games stock a.i. bots would come under the definition of a mod.

Modding is described online as…

(R, 2001)“"Modding" is just jargon for "modifying" – altering – video games.

Savvy fans dive into the back-end of their favourite games to fix bugs,

update graphics or introduce new elements.

Sometimes, fans create new games altogether”

By Business Insider, and this for me it’s a noble art and why I have chosen to do this toolkit, but I’m not only doing this for the propose of changing and bringing new life to a game that’s been around since 2004.

I hope that using this will be a gateway for more people to get involved with modding video games. This is because the environment for modding is getting far stronger day by day, and with the support given to casual gamers to install mods.

e.g. Steam Workshop and gametechmods.

So, the community has embraced the idea of mods, and many people enjoy them. Even to the point of YouTubers endorsing and showcasing mods frequently like in the example video ((2012), 2017)

So, it’s very popular with people for use but not so much with creating them. This is as I stated earlier not many people know how to code, or how complex code works. This is required for modding, the main way of creating these now in the modern day is reading code and using a multiple number of custom tools. My goal is in making one tool that is well documented and takes on almost all the tasks required to creating a mod, then it should remove the barrier of entry into this fun hobby, and maybe into game development as a whole.

The game I have decided to create this toolkit for is the 2004 release Robot Arena 2, made by Infogrames.

The game idea is very simple, you create a fighting robot from scratch, this includes many a thing such as:

* Creating the chaise,
* Adding custom made using individual parts (e.g. making an axe out of a pole and ice pick head.),
* Wiring up the controller for the mechanisms you’ve created,
* Painting and making the bot look its best.

Below is an example of a bot I have created myself for testing with this project;



"This an example of a bot I have created myself for testing with this project;"

Once this a bot has been made you can fight it against A.I. stock bots in many tournaments or single fights, you can also fight against over players.

The game is on the developers site and described as such:

(Gabriel Interactive, 2017)”Robot Arena, the first ever robot combat simulator, was such a popular product that Infogrammes published the sequel, “RAII: Design & Destroy.” Enhanced collision, greater combat realism, and bot workshops empowers players to build truly unique robots and take them into online multi-player battles.”

This game is very well studied by an army of fans who has documented how the game works in detail and how to change them. One of these groups of people that assist in fixing, modding, and documenting the game as well as sharing custom bots is GameTecMods. These guys main way of commutating is via the forums.

Here’s an example of one linked (GameTechMods, 2017)<https://gametechmods.com/forums/modifications/complete-robot-arena-2-mod-index-wip/>

There are also mods created for the game that completely overall the entire game, so known to be very mod able.

This makes this game a very good title to try and create a tool for.

Project Objectives

This project has a few main objectives that I am aiming to hit by the end of this project and they are…

* Having a GUI that writes the code for the bindinngs.py file need to add a new computer-controlled bot into the game.
* To change the games stock a.i. code to allow computer-controlled by the a.i that attack backwards as well as forwards.
* Improve the overall quality of the code going into games a.i controllers.
* Set up a version of the games files so that new people to the modding community have the base to modify the games files outside of the a.i of the game. Including more advanced features like custom boundaries.

Timescale

The first task is to create a bot in game and get it to fight by itself,

Create a visual interface that can create a new line with-in the bindings.py file. This could be the longest task as it after this everything else functionally wise can just be added in as a new file.

After that in detail reading into the a.i. files in the attempt to create a new file for the test bot created in the introduction.

Next, I can then create a method of writing a nice new user inter face that takes the type of bot the user wants and make the code needed to get the bot the user has created in game and add it as a computer-controlled bot.

Finally, I can add in code into the super ai classes in the games files that can add in the extra functionality that is needed to make the bots attack backwards when they lose all their weapons. Then I can investigate a method of attacking when the bot only loses one of his weapons.

After the tasks above have been completed then I can look into furthering the improvements in the games default a.i. where i can find faults.

Statement of Originality

**CS3D660 Individual Project**

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerized sources have been fully referenced, and I fully realize the consequences of plagiarizing any of these sources.

Student Name (Printed) Jonathan G Watson

Student Signature jwatson

Registered Course of Study Computer Games Development

Date of Signing 19/04/18

# Table of Contents

Contents

[Title Page 1](#_Toc511941050)

[Project Title: Modding Toolkit For Robot Arena 2 1](#_Toc511941051)

[Table of Contents 6](#_Toc511941052)

[Background 6](#_Toc511941053)

[Design 13](#_Toc511941054)

[Methodology 15](#_Toc511941055)

[Results 25](#_Toc511941056)

[References 25](#_Toc511941057)

[Ethical, Legal, Social and Professional Issues. 25](#_Toc511941058)

[Appendix 25](#_Toc511941059)

# Background

Control of weapons

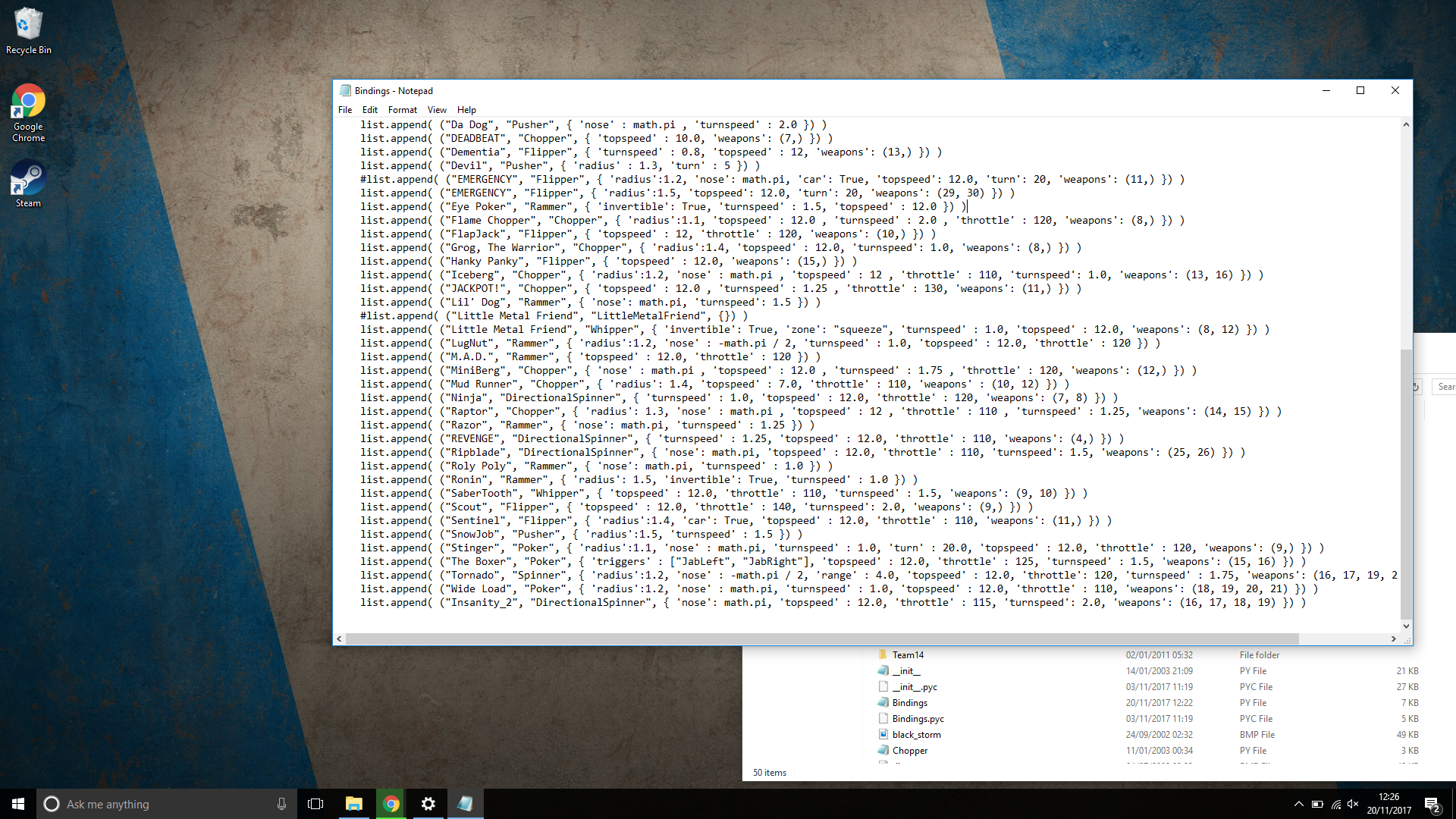
If you have any other type of bot with an active weapon then you need to be able to know when an enemy bot is in the strike area of the weapon, to be able to get an effective hit in. so you need to be able to map area in front of your bot’s weapon. So in Robot Arena 2, this is simple it’s called a “Smart Zone”, this in the game is a part that is hidden to the player, it’s a simple 3d box or circle that is invisible to the player, so if another bot enters the smart zone it triggers, the function called smartZones in the \_\_init\_\_.py file (I will go into more later in this document). This method is used by robot models that use weapons like axes, flippers, crushers and pokers. [2]



[2] the image above has two examples of smartzone, you can see it in my game because I have moded the components file register in the games files, so I could make the smartzones visable as these orange/brown rectangles.

Bindings.py

Outside of the way the. bot files and in game components, how does the A.I. get blinded to the bots in the game. It’s done though a list that is read into a registry that is passed into the \_\_init\_\_.py file. This file that holds and passes this information is called “bindings.py”. I have screenshot my messing with this file [3]



[3]This is the bindings.py file after I had created a binding for my bot “Insanity 2”.

Well first I created the bot “Insanity 2” in the game as normal, this bot is what is known as a directional spinner, this is a bot with a spinning weapon that can only strike in a set area of the bot. so this bot has no smartzone as it’s a spinner, so how does the game set out a bot like this?

Well in the case of Insanity 2 here is the snippet of code from the bindings.py that is used for Insanity 2:

“list.append( ("Insanity\_2", "Spinner", { 'nose': math.pi, 'topspeed' : 4.0, 'throttle' : 110, 'turnspeed': 2.5, 'weapons': (17,16) }) “

The first of these commands is the “Insanity\_2”, this is the name in the .bot file that the A.I. will look for to know which bot is it trying to control.

Next is the weapon type of bot the named bot is “DirectionalSpinner” in the code this looks for the SuperAI file to use for the bot.

the rest is simply the speeds that it can achieve, how fast it can reach its top speed, how fast it can turn, what components are used to create the weapon/weapons on the bot.

\_\_init\_\_.py

The main way the game deals with path finding is done though a c++ file that I will not be changing as it works perfectly well.

The way that the bot’s choices are made is through a controller class that is called \_\_init\_\_.py. this file deals with the where a bot is heading for, which of the other bots in the match to target, if the bot is alive or not, and lastly loading in the superai files that needed for the computer-controller bots.

Also worth mentioning is that the bots firing of weapons are the SmartZone’s controlling them are all dealt with in this file which means that, in our later tactics files we don’t have to worry about weapon controls.

Tactics.py

The next files is the tactics file, this is used to create the basic outline of what all the superai basic functions should be. So is the parent class to all of the superai classes in the game, so the main things that are done here are what all the bots in the game need.

These things include what to do when playing in different types of matches like for example the function for ‘Reign’ that is used for making the bots stay in a scoring zone.

class Reign(AI.Tactic):  
 name = "Reign"  
   
 def Evaluate(self):  
 self.priority = -1000  
   
 # we always want to be king!  
 self.priority = 100  
   
 #~ # if we're not in the scoring zone, let's go for it  
 #~ a = Arenas.currentArena  
 #~ s = a.GetScoringPlayer()  
 #~ if s is None or s != self.ai.GetID():  
 #~ self.priority = 100  
   
 def Execute(self):  
 # occupy the scoring zone (and/or shove close robots out of the way if we're in there)  
 a = Arenas.currentArena  
   
 loc = a.GetScoringLocation()  
 dist\_threshold = 2  
 push\_threshold = .3  
   
 reverse = False  
 dist = self.ai.GetDistanceTo(loc)  
 h = self.ai.GetHeadingTo(loc, False)  
 if h > math.pi / 2 or h < -math.pi / 2:  
 reverse = True  
   
 if dist > dist\_threshold:  
 self.ai.DriveToLocation(loc, reverse)  
 else:  
 # slow down when we get there  
 speed = self.ai.GetSpeed()  
 if speed > 2.0:  
 self.ai.Throttle(-100)  
 elif speed < -2.0:  
 self.ai.Throttle(100)  
 else:  
 # keep facing the nearest enemy & ram him if he gets close enough  
 self.ai.Throttle(0)  
 id, dist = self.ai.GetNearestEnemy()  
 if id is not None:  
 h = self.ai.GetHeadingToID(id, False)  
 if abs(h) > push\_threshold:  
 self.ai.AimToHeading(h)  
 elif dist < 4:  
 # push him full power  
 self.ai.Throttle(100)  
   
 return True

the way that the bots tactics are run though is a method of creating classes that the \_\_init\_ file binds to the bot whilst the game is running.

All the classes that can be used has a basic structure that I can outline as

First ‘name’, this is where we tell the bindings file which class is being used when going into the bindings file usually.

Next we need to run the function ‘Evaluate’ in here we set the priority of the bot, and what other things we made need to consider special conditions that would differ from the normal functions of a bot fighting, in the example class we are looking at it would be the arena, who the current scoring player is.

Next the ‘Execute’ function deals with the anything we need to do after every tick of running in this class. So in this example its move towards the centre of the scoring zone, then based on that location whether or not to reverse or go forwards.

In this example then after we know our location and if we are going forwards or backwards we then we find the nearest bot to this bot and face it. If he is nearer than 4 units then we will turn on the throttle to 100 to push the enemy out of the zone. If there is no enemy in the zone, then the throttle is turned to 0.

SuperAi (named after weapons.py)

These files are the ones that are for how the bots should be loading into match and how they will fight if it’s just the bots are plain fighting with all components attached.

The same as with the tactics file there is an outline for each different iteration of this file type. This outline is shown using the simplest type of SuperAi file that I know of in the games file. This is the Rammer.py file…



So, on here I will so what the main things needed are for a SuperAi file to work,

The first is the registering the bot to the \_\_init\_\_ file though inertance we can use that file and add any other values we may need to add into the file for this type of bot only. In this example none are needed so we just run the wanted class from within the tactics file.



The second part is running the activate function. This function tells the bot where to face and to evaluate itself and the other bots are around it.

We also have some debugging code that prints out where the bot is looking when he is activated in the match.



Lost component is next part of the code and all this function is used for to report back what parts are broken off when they are, and what action to take for that happening (on almost all the bots they become pushers when all weapons are lost)



And then finally a debugging function that outputs where the bot is looking at that set time.



Inheritance and file structure

A close up of a logo

Description generated with very high confidence[4]

The diagram above [4], shows the method from my research of the game I believe is used to pass all the functions to the SuperAi that are required for the AI to run from the \_\_init\_\_.py and tactics.py files.

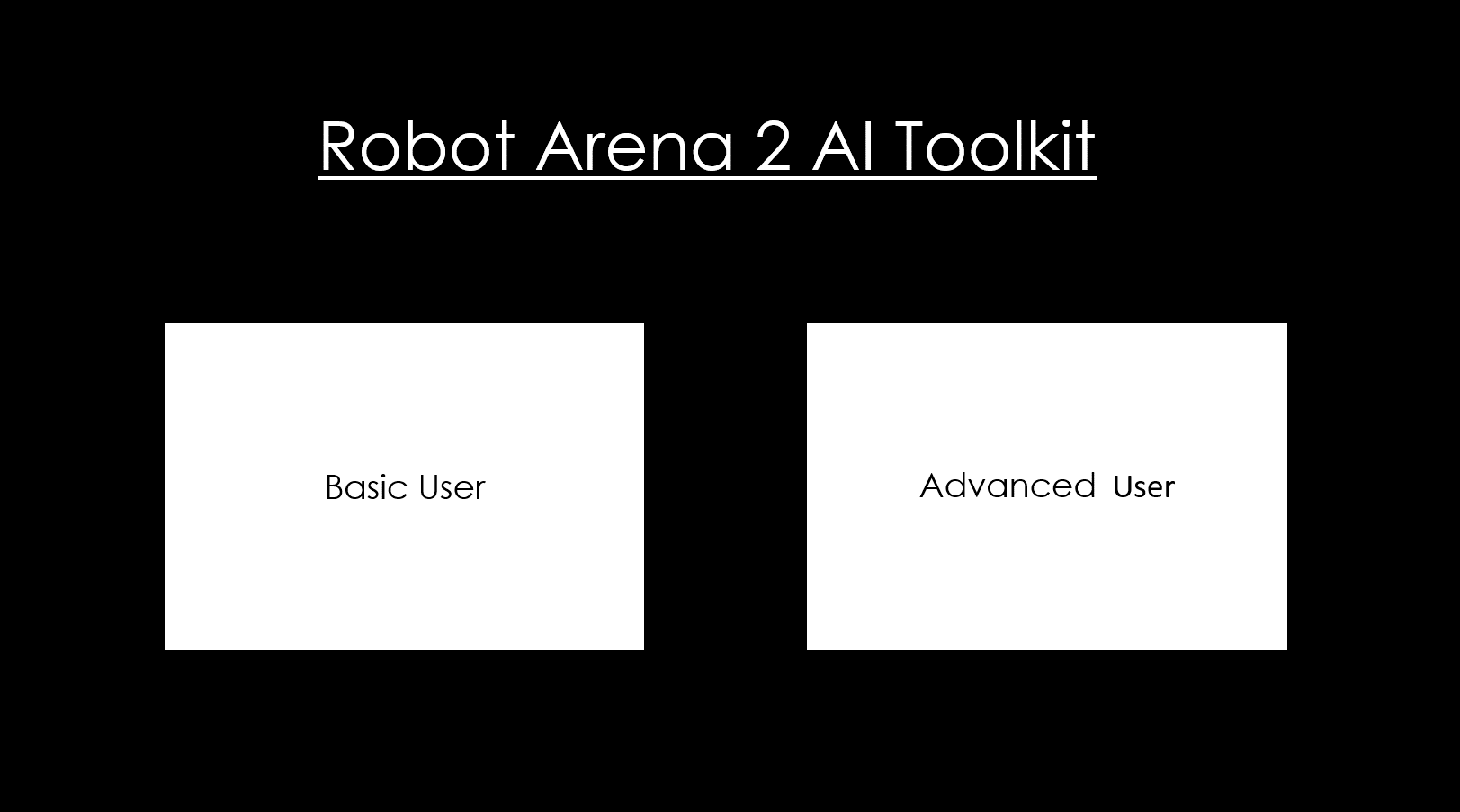
# Design

User interface

This is the part of the software that the end use will see and use. Therefore because of the users I wish to have using this software, it must be easy to understand. So, this to me means being minimalistic but clear about what you are filling in without jargon and appealing to the eye.

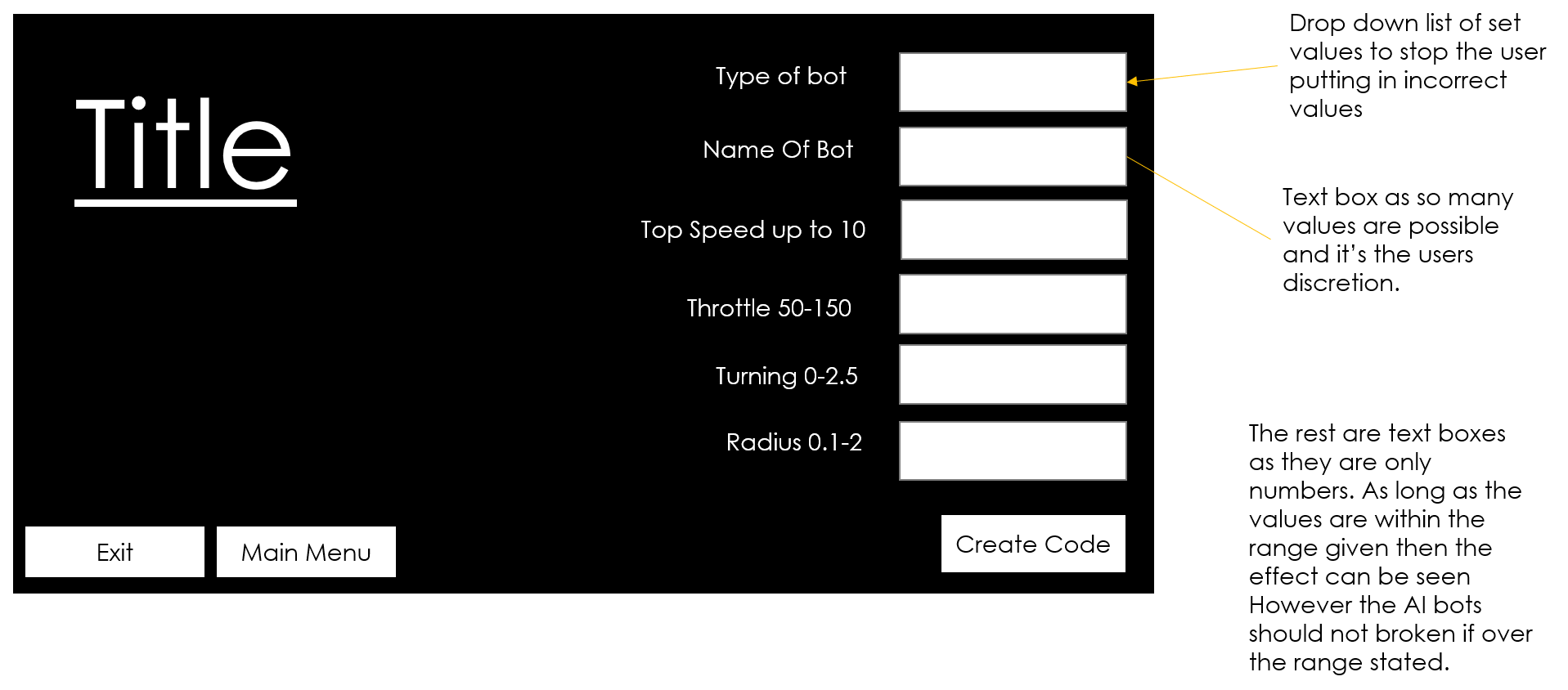
I also wish to create a more advanced version of the tool to allow slightly more advanced users the ability to adjust values like how many weapons and where they are located.

Home screen design



This is the page the user will see when they start the application, therefore it is simple and not overbearing to the user. The page should be appealing to the user, having a minimalistic style makes it look modern and simplistic. Overall this page does what I this page need to do no more and no less.

The Function page



As with the home page this page is minimalistic in style and simple to understand. It is a bit more filled in but to the eye of a user is not scary and for a first time Modder. I believe that having labels with ranges makes it easy for the user to understand without a huge amount of text being on the page. This keeps the style of the application but also makes sure the user knows the what they can put into the boxes. The only box that doesn’t use a text box will use a drop down box with clear labels of the bot types that the user can select from, this will reduce confusion for the end user.

The layout of the drop box would be as follows…

|  |
| --- |
| Flipper |
| Axe Bot |
| Directional Spinner |
| Spinner |
| Rammer |
| Piston |
| My own type that I haven’t named yet |

# Methodology

SmartZones

The first thing I did was make it so that the user had access to the SmartZone’s components within the game itself. This was done first extracting the compressed files with in the components file.

This was done using a tool called “uncfl” this can be found at (GameTechMods, 2017)<https://code.google.com/archive/p/gmftoolkit/downloads>

. To use this I simply had to drag the folder I wanted extracted onto the uncfl application.

A close up of a device

Description generated with high confidence

“Components” Before folder before being extracted

The extraction of this folder allows for a user like myself to start changing how components work, though things like the weight, the amount of power they use/give and most importantly if they are visible or not to the user in the games bot workshop.

A screenshot of a computer

Description generated with high confidence

The components folder after using the “uncfl” tool. Showing all the attributes for each component in the game.

The task of making the SmartZone appear in the bot workshop was simply to turn the value of hidden to “0” instead of “1”.

A screenshot of a cell phone

Description generated with very high confidence

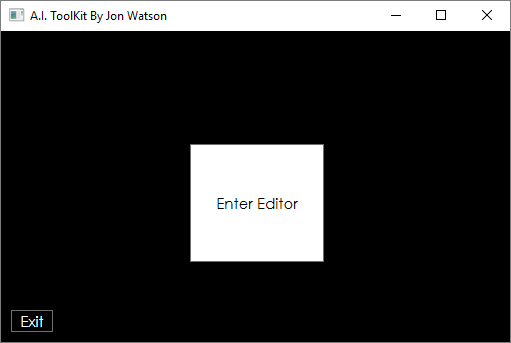
The smartzone file being made visible to the user by changing the “hidden” value

GUI Interface

The first weeks of creating the project was spent creating the interface for the system, I have used visual code is using XAML whilst the background logistical language is C# for the GUI interface. my main reason for using this setup for my GUI is that this type of interface was created by Microsoft for this type of application. The XAML is relativity simple to use and it is a very well documented language online I have found in the last few weeks.

The program I have working now works for the vanilla games A.I. I have made this way because when I add my own A.I files into the game they will still need to be bonded to the bot though the bindings.py file but I will call my own A.I tactics files in instead of the vanilla games files. They can even use the same option names as the ones in the vanilla game, so once this done I don’t have to adjust it much if at all after making it.

The GUI interface is simply supposed to ask the user a few basic details of what type of bot they want A.I. applied to, creating the buttons and the visuals of the GUI was done quite quickly and I have screenshot them below



[3] The main menu page the user first see as they run the GUI.

Above is the first page the user will see when they run the application at the current time its made using two simple button that have events that when clicked opens the relevant page and then closes the current one, here is an example of what one the button to move to the basic user looks like:

private void Basic\_Button\_Clicked(object sender, System.Windows.RoutedEventArgs e)

{

Window Basic = new Basic();

Basic.Show();

this.Close();

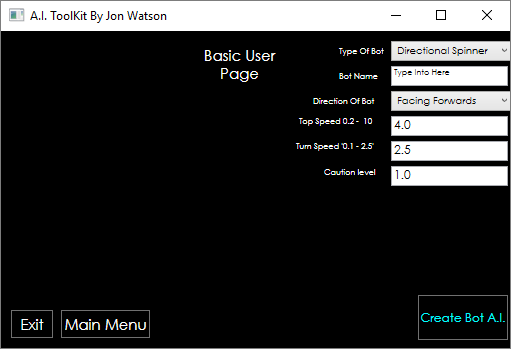
}

In this code first line after declaring the function simply creates a new instance of the window “Basic” that is called “Basic”, when this happens all the variables are created as blank.

The next line then simply displays the new window we created. “Basic.Show();”.

And finally, in the final line we close the window we were just on.

This code is the same across all the navigation buttons I have in the program now, so if we use the basic button as shown in the main menu screenshot [3] then we will come to the actual tool itself shown below:



[4] The basic user page filled in for Insantiy\_2.

Here we see a few options so I will go through what each does, the problems I have with implementing them and how they can be improved. To show this I will show how a user would use the toolkit to A.I Instanity2 the bot I stated I would use as a case study in the introduction [2] :

* Navigation Buttons:

I stated in the last paragraph these are the same as the buttons on the main page and are working well so can’t be vastly improved upon.

* Type of bot:

This is the first option at the top of the page the user should fill in first, this is a simple drop-down menu the user uses to select the description they would say closest matches their bot. as an example of this working if I was to do this for Insanity\_2 then I would say my bot is closest to a Directional Spinner.

So that’s what I would select (it is also the default option).

In the code each option has an event handler on it and this has a function on it that sets a string to be printed into the bindings.py file that describes the type of bot we are using. Here is the example of the Directional Spinners event handler:

private void Directional\_Spinner\_Selected(object sender, RoutedEventArgs e)

{

bindings = "\"DirectionalSpinner\"";

}

In the constructor of the code the string “bindings” is created blank.

In the event handler this “bindings” string is set to match the format of the bindings.py file so for the Directional Spinner type bot this string would read “DirectionalSpinner”.

The hardest part of this part of the code was getting the speech marks to be stored as part of the string, the solution to this is the special input method. That’s why the string value looks a bit weird the \” and the \” are exception handlers that can record the speech marks as strings. This took me a fair amount of time to find out afterwards I found it online in a help site called stack over load (CodeGuru, 2012). This reduces the number of things I must do to get a full line printed into the bindings.py that will A.I the user’s bot using the default games A.I.

The only way I think this could be improved is in the code readability. As currently it is quite hard to follow what the codes doing, and when a function runs or not. Now due to the comments you can tell but in future I may use Boolean tags to set the bot type, then use one big function to set all the strings into one variable together before the printing stage, but that is for readability only, as is the code works and is reliable.

* Bot name:

In this text bot the user enters the name of the bot as it is in the. bot file. Or another way of putting it the user puts in the name of the bot as they did when they named the robot when exporting the bot from robot arena 2, so in the case of my bot this would-be Insanity\_2, as shown in the example image [4]. Whatever the user enters is stored as a string after being formatted to match the python script in the bindings.py file.

The event handler is triggered when the user enters new text into the text box in the window as this means the string is always up to date with what the user has entered it as soon as they have finished typing it. This gives me high accuracy.

Here is the function that passes this text into a string formatted

private void bot\_name\_TextChanged(object sender, TextChangedEventArgs e)

{

bot\_name = System.String.Concat ("\"", bot\_name\_button.Text, "\"",",");

//system.string.concat adds two strings and makes them into one value

}

The “system.string.concat” is a function that I found on the stackoverflow site (CodeGuru, 2012), this adds the stings passed to it into one string.

* Direction of bot:

This tells the game which way the bot should face when it is entered a match. This will almost all time be forward or in the bindings.py file this would be “'nose' : math.pi” the dropdown menu used here works in the same way as the bot\_type drop-down did instead though this one prints out the maths method required to get to bot facing forward when spawned into the game.

So here are the four options I have put into the menus print out as in the binding.py and are therefore stored as in the stack:

1. ‘forward’- "'nose' = math.pi"
2. ‘backwards’ - "'-nose' = math.pi\*2";
3. ‘left’ - "'nose' = math.pi/2";
4. ‘right’ - "'nose' = -math.pi/2,";

And here is an example of the event handler function for the faceward option:

private void Facing\_Right\_Selected(object sender, RoutedEventArgs e)

{

direction = "'nose' = -math.pi/2,";

}

The main way of improvement on in for a more advanced user by allowing to enter an exact angle to face the bot in, but for new basic user it should work well enough.

* Topspeed, Turnspeed and Radius:

The last three are all textboxes for now using the same method of updating their respective strings. They control the way the bot is driven, the “Topspeed” states the topspeed the bot will ever attempt to reach, the “turnspeed” states the max rate the bot will try to turn and the box labled “caution level” stores the radius value, this dictates the size of the circle used by bots A.I to look for hazards, if the value is low then the bot is very aggressive but will hit hazards that could disable it. The higher value bots are less direct but tend to be very good at dogging the arenas hazards.

The update functions for all three of these event handlers are shown In full here

private void TopSpeed\_TextChanged(object sender, TextChangedEventArgs e)

{

topSpeed = System.String.Concat("'topspeed': ", TopSpeed.Text , ","); ;

}

private void TurnSpeed\_TextChanged(object sender, TextChangedEventArgs e)

{

turnSpeed = System.String.Concat("'turnspeed': ", TurnSpeed.Text , ", ");

}

private void RadiusBox\_TextChanged(object sender, TextChangedEventArgs e)

{

radius = System.String.Concat("'radius': ", RadiusBox.Text);

}

Again, using concat to join the values to the format required to be used in the bindings.py file.

In my Insanity\_2 example the bot was made it be fast and cautions of what’s around it so the values I filled in are topspeed: 5, turnspeed 3, and caution level of 2.

* And Create bot A.I:

This this the last of the buttons in this tool and is the last thing the user should hit after filling in the entire form, this button takes all the strings in order adds them into one single string and prints it into a text file (this will be the binding.py file after being tested.) and then creates a dialog box that informs the user their binding has been written to the bindings.py

Here is the code for this buttons event handler:

private void Save\_clicked(object sender, RoutedEventArgs e)

{

//adds the all the varibles together as one string to be printed into the bindings.py file

final\_printout = System.String.Concat("list.append ( (" , bot\_name, bindings, "\"" ,", {" , direction , topSpeed , turnSpeed, radius ,"} ) )") ;// feeding the string to the text file

System.IO.File.WriteAllText(@"..\..\..\..\testdoc.txt", final\_printout);//writing the document. currently writing into testdoc.txt needs to be changed to bindings.py before full release.

//file path needs to be changed before publishing.

MessageBox.Show("Bot written to bindings.py");//printing out a message to let the user know that the code has written

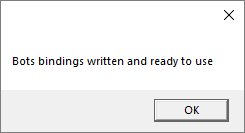
}

So, the first part of this is using the concat function so works the same as the other input but, it is taking in all the other strings in and making them into a string called “final\_output”.

In the nextline we print this string into a text file (in this case called testdoc).

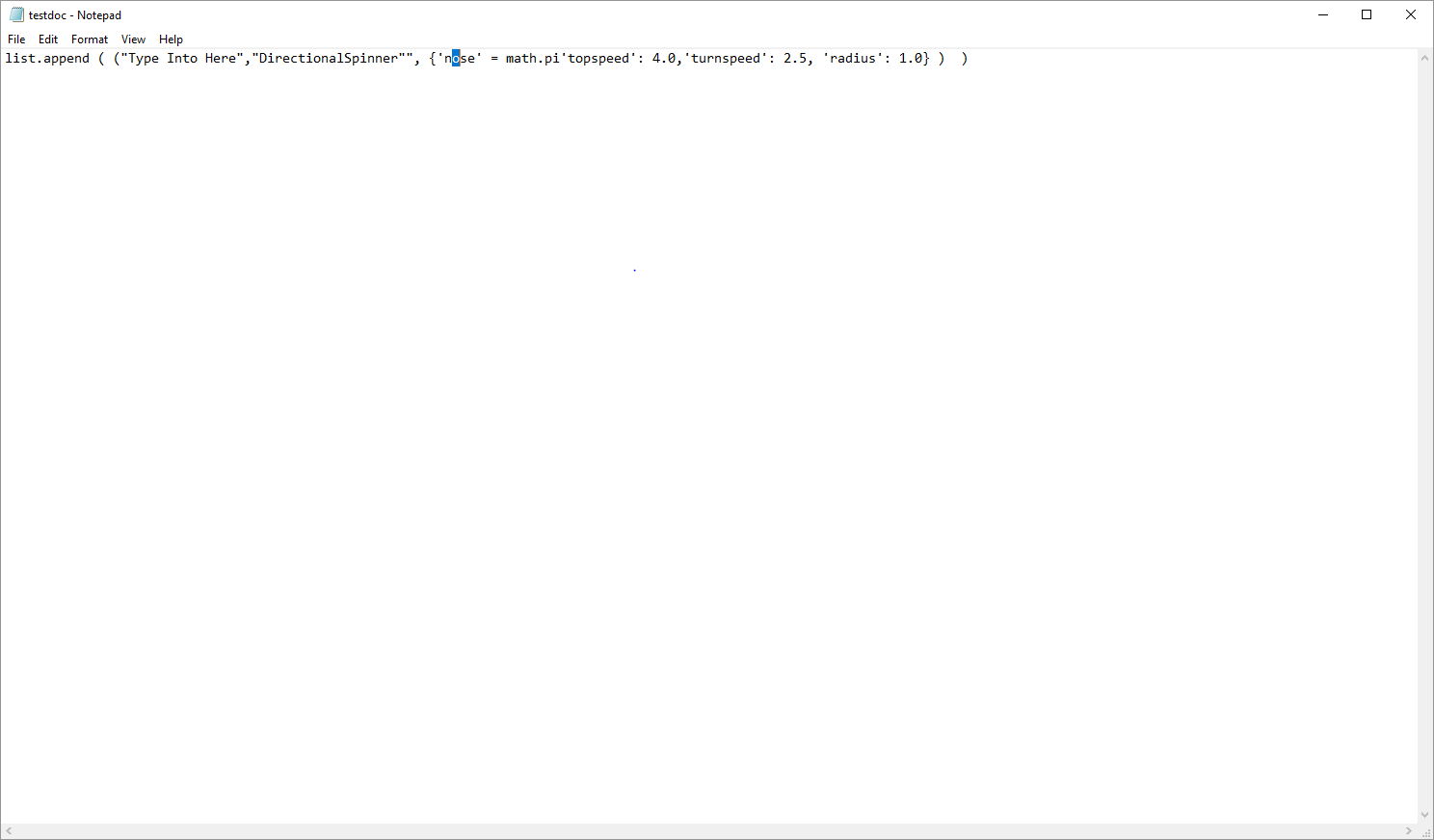
And finally, a window, telling the user the line has been done.[6]

[6]screenshot showing display window



And here is the testdox.txt file after we filled it in for insanity\_2[7]

[7]screenshot showing testdoc.txt



JonsBot and modifications to the \_\_init\_\_.py file

The main medication to the default A.I. was to make it so that the bots could attack backwards as well as forwards.

I have used my test bot ‘insanity\_2’ as the way locating the best method of achieving this goal. This bot was a spinner, so I started with spinner SuperAi class. This class only attacked in one direction. Plus, if all the weapons in the list of weapons loaded in when the bot’s variables are taken in by \_\_init\_\_ are removed then the bot changes into a rammer/pusher.

Some of the functions in this class are very useful for the feature I wished to add to this AI class, for example, every tick if another bot is in the within the range we set for the bot’s weapon then they will turn on, otherwise they turn off. This type of function can still be used and would be more than helpful in the case of making a bot attack both forwards and backwards, this is because this way I don’t have to worry about the weapons working when running backwards, if an enemy is in the circle of range then the weapons will be on.

Therefore I only need to worry about what direction the bot is driving in is based on what weapon is currnetly attached. If the first weapon is atteached then the normal course can be taken to contoll which way the bot is orianted, if the first is broken and the second weapon is still attached then we switch the second weapons smartzone as the front of bot. This also works for axe and flipper types bots, these types of bots require their smartzones to be able to be fired. In the shot of code below is the simple if statements that fire weapons in this case.

A screenshot of a computer

Description generated with very high confidence

And we also need a way of getting the smartzones into the superAI so at the setup of this bot we take the extra weapons added in the bindings required and add then to the bot in separate triggers for these weapons.

A screenshot of a computer

Description generated with very high confidence

Apart from this I also added in a sweapons list into the \_\_init\_\_.py file again so that we can check which of our weapons is still on one by one using Boolean statements.

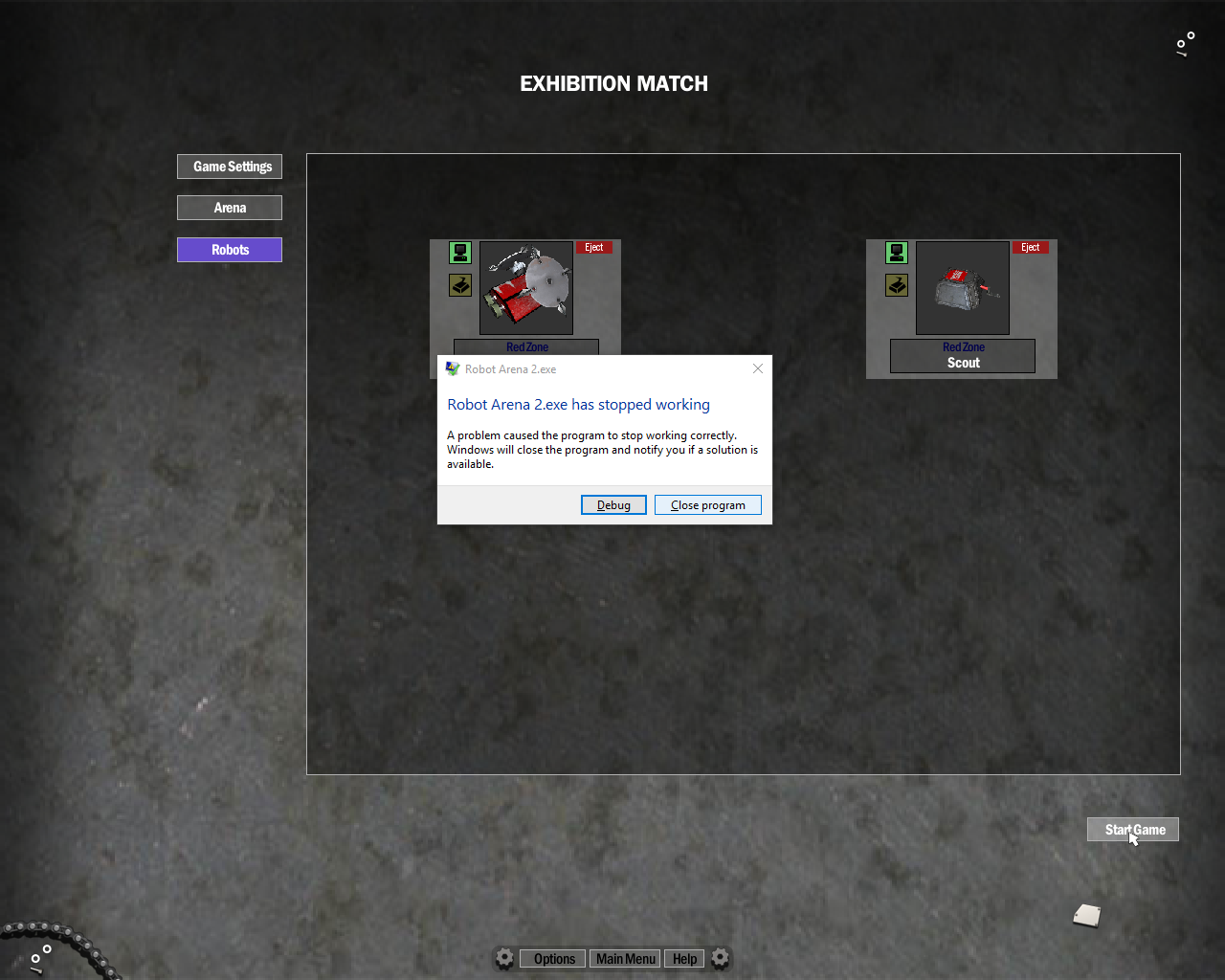
A screenshot of a cell phone

Description generated with very high confidence

# Results

Overall

Overall the toolkit part of the project works perfectly, when I placed in my first test bot that I used as a testbed for creating the toolkit and research worked perfectly with the toolkit. However, with the actual modifications to the core A.I. pack that I created “JonsBot” the SuperAI class I created I am unable to tell if the code works correctly or not. This I because when I ran and developed the code with the operating system windows 10. Other AI packs use older versions of windows that the game was originally built with. The end effect is that the game stops when the JonsBot runs with more than one active weapon system is loaded into the game with windows reporting the game stopped unexpectedly with the following window



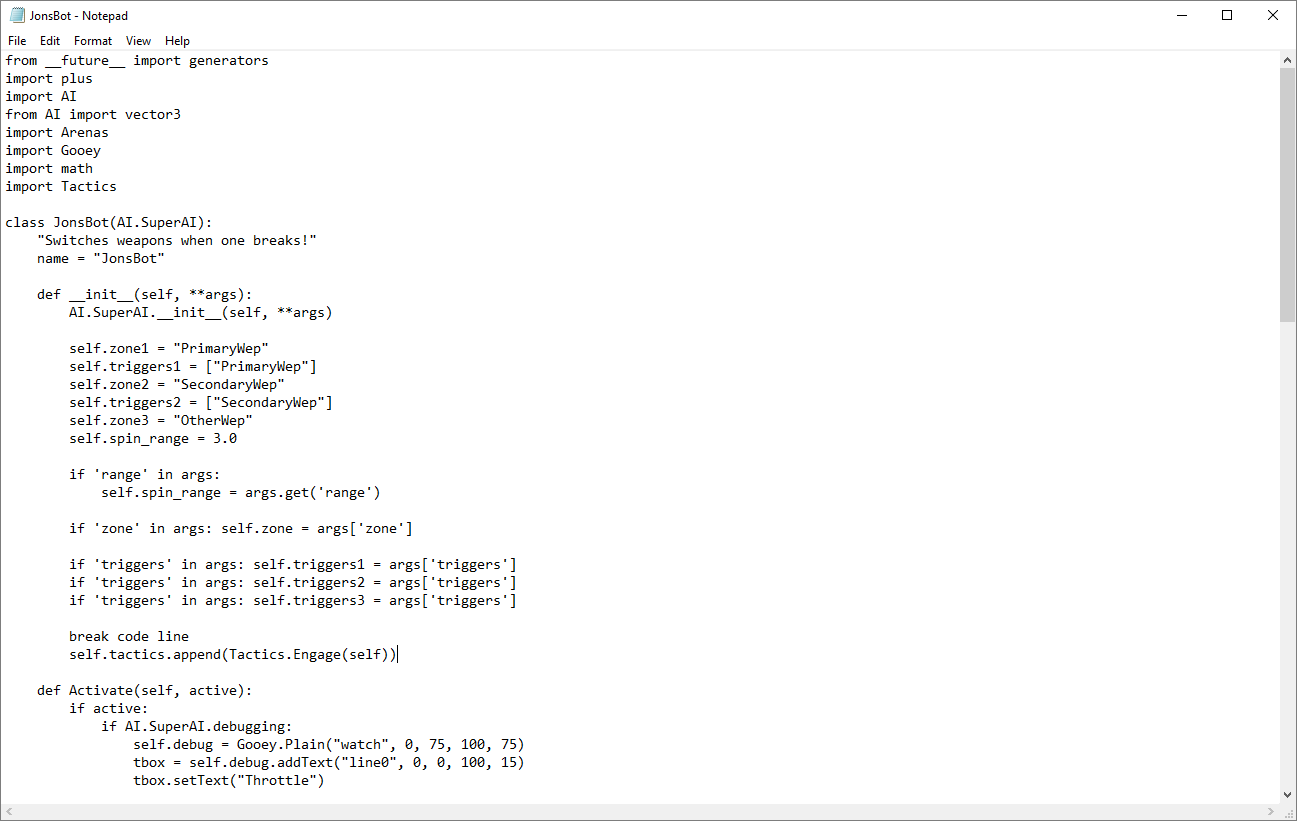
Error message that shows when the game runs the JonsBot SuperAI with two weapons systems on.

A screenshot of a social media post

Description generated with very high confidence

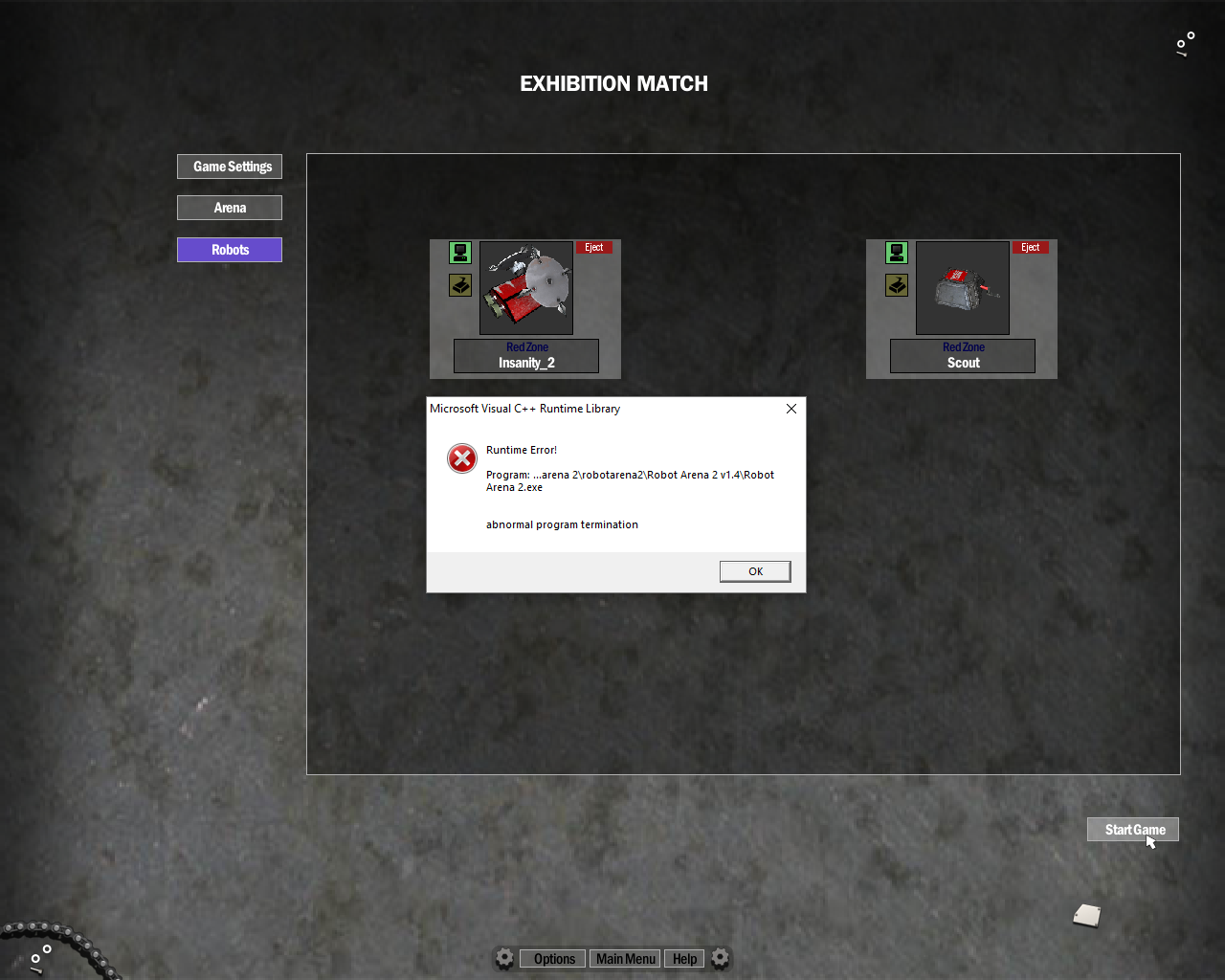
Here is the bindings file showing the robot “Insanity\_2” with two active weapons systems

The main reason that I can tell this isn’t an issue with the OS and not the code is that the code compiles. If I purposely put in a line to break my code, then we get an error from the game that tells us the game has crashed due to improper code.



Above the \_\_init\_\_ function in “JonsBot” that will break the code dude to me adding the line “break code line”. This will break the code as it there is no syntax in that line of code.

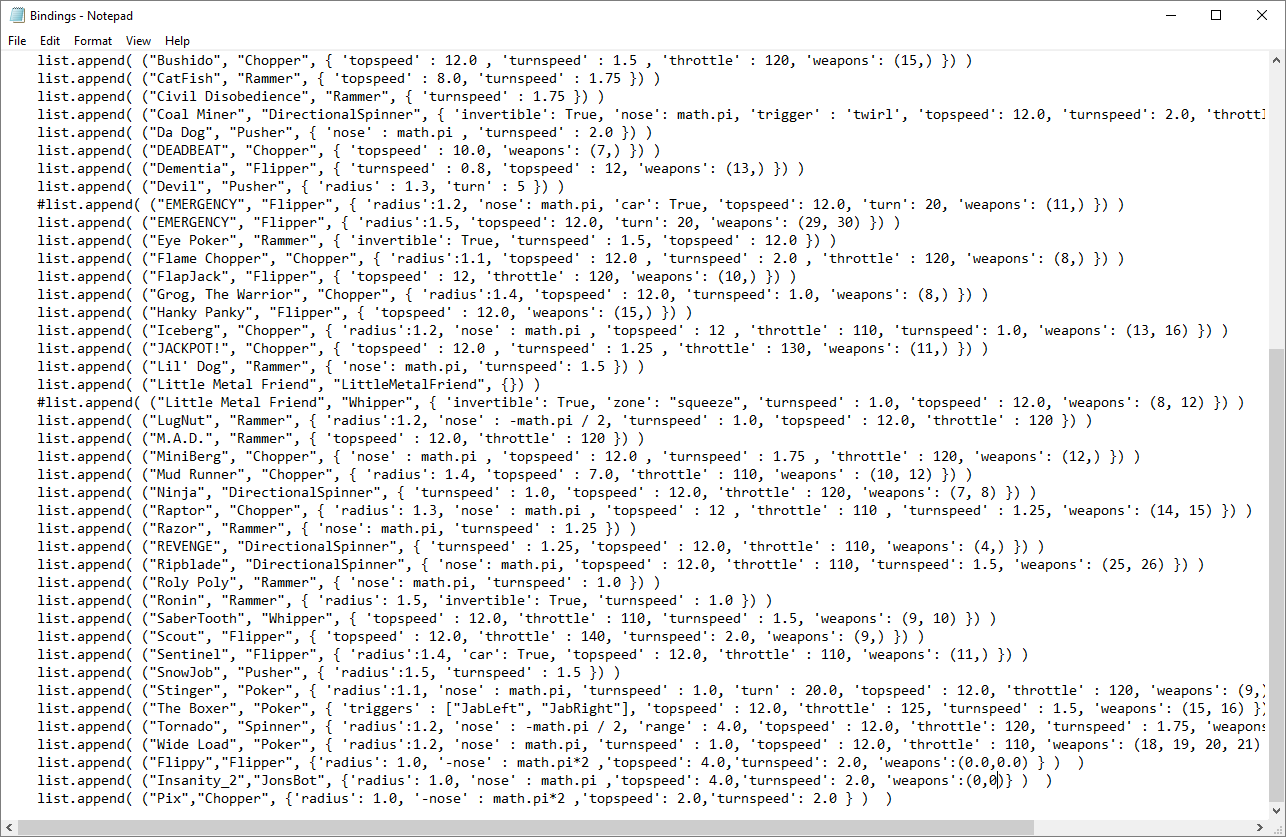
So I ran the game using the code above and the game stopped but with this error instead

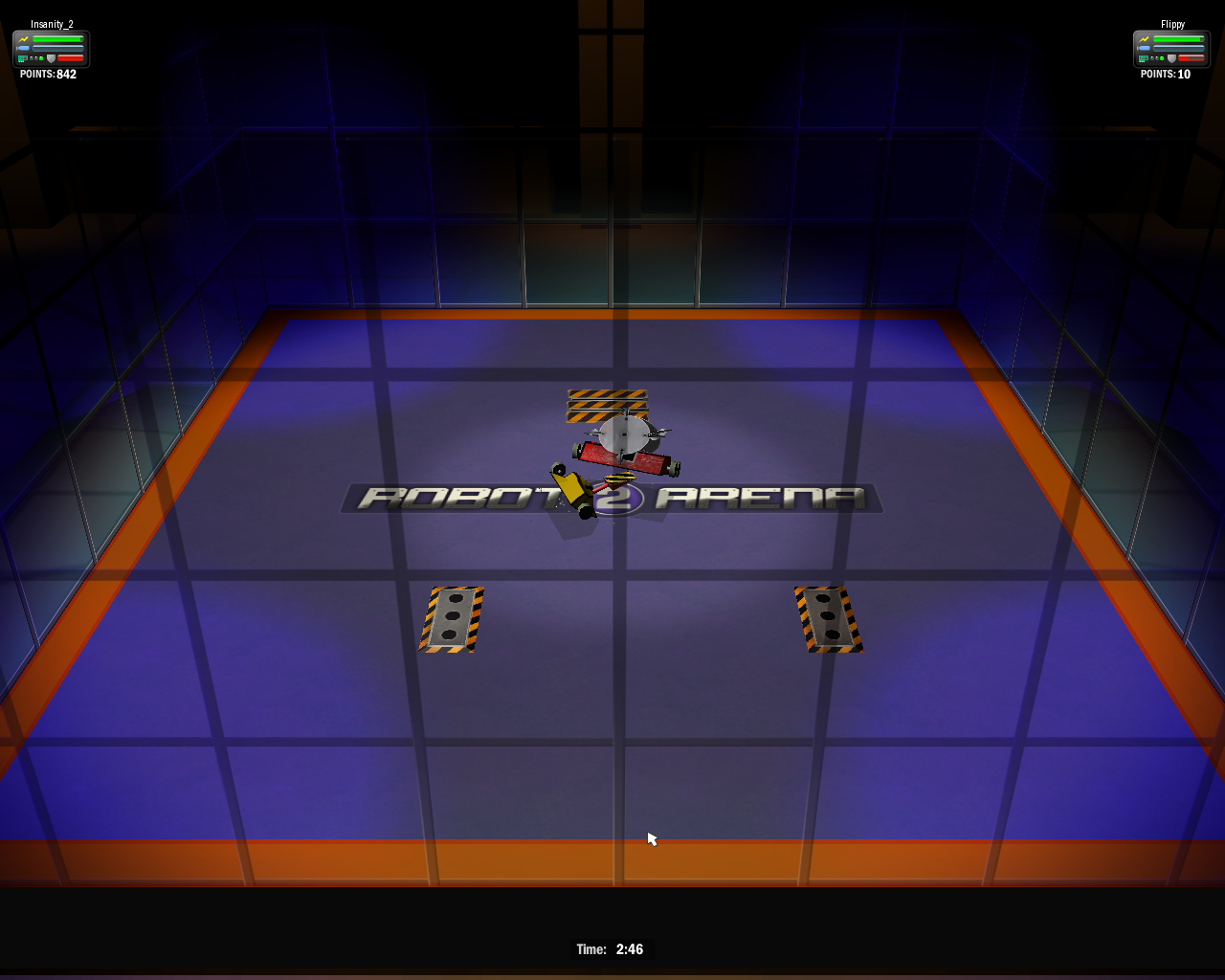


The game crash message when using code that runs incorrectly.

The above screenshots tell us that the OS is stopping the code from running with two weapons systems, this is as the code is compiling and running but the game is being stopped by the OS. I have also tried to run the essential AI code by the BeetleBros (BeetleBros, 2018) at <http://beetlebros.gametechmods.com/robotarena.htm> and it also had the same error. They are known to be very reliable code makers that used an older version of windows, therefore I believe that its is the OS not my code itself.

This is also shown by the fact that I remove the second weapons list then the code runs perfectly. When Running the bot appears to use the back weapons more than when the bot was using the original “Spinner” SuperAI class.

”Insanity\_2”’s bindings without the second active weapons binding



Snippet from video showing the AI fighting, showing both custom test bots fighting full running with their own AI.

In summary the SuperAI class I created seems to be more aggressive and uses the back disk more than the original “Spinner” class that I modified to create the “JonsBot” SuperAI. So overall, I would say its achieves both goals that I set out to achieve with this part of the project. Though I can’t be certain about how well the bot attacks backwards.

The final issue I noticed is that with some of the “axe bot” style bots they’re weapons didn’t fire. I believe that this is due to the bot not knowing which part was the motor responsible for this action. All other bot types seemed to work with very little to no issues and the Smartzones mod I added in worked without an issue ether.

User feedback

Overall though the toolkit does the job I wanted it to do according to my users of the code

I asked some of the users for general feedback on how the toolkit worked and if they thought it made modding easier to do/understand, here is their feedback:

This project has a few main objectives that I am aiming to hit by the end of this project and they are:

Georgie Chard (First Time User):

From never experiencing game modding, I managed to create a little bot named Alex using what was provided, most of which I found easy to use and understand.  
When setting up the bot, I thought the process was simple but also challenging as you had to come out and go back into some parts, which can be confusing if you’re not so familiar with the setup of the game e.g. saving your design as there is no save button.  
Using the toolkit was a little too basic as when choosing which type of bot you’d like, there’s no description telling you exactly what the feature entails but instead, gives a short, sharp option like you’ve played this many times. However, the rest of the sectors were very clear.  
Using the product was quick as it was basic copy and pasting, and the actual code itself was nothing too complicated, so I understood completely.  
I noticed the differences immediately when in gameplay with Insanity\_2 as this bot was more aggressive and assertive, exactly like how you’d want your creation to behave. Whereas with Alex, you can tell that it hasn’t had that much time put into it and how it lacks in performance.  
Overall, I found the experience of using this software helpful and very easy to learn by. The only improvement I could suggest would be the organisation of the files. They were scattered around and hard to locate when needed, it would’ve been better to have them all in one place.

Daniel J Harris (Alert Logic - software engineer):

When setting the bot up in the game for the AI kit via the instructions given to me, I found the task easy enough to use, had a little trouble using the track pad, don’t think this would have been an issue if I used a mouse. I liked how integrated the Smartzone felt, just like using any other accessory.

When using toolkit, I found overall it was very simple to use and understand, it is very basic but does not need to be any more complex for what it is doing in my opinion.

I found using the line of code that was created a little tedious because this needs to be done every time, but the process was simple enough.

It was very entertaining to see my design fight, and because its controlled by the AI I don’t feel as much to blame when it loses, as I know the AI would fight in the best possible way. If I were playing myself I would blame myself for loosing. Especially when I was going up against the advanced bot, it seemed to be better equipped with the double blades and it had an easy job at hunting me down and killing me. Moving the files and text around is very clumsy, would be much better if this was done automatically. But I found watching my design fight very entertaining, I would love to see a betting system integrated into the fighting, that way each person could put in their best design and not have to worry about piloting the bot.

The interface to create the code worked flawlessly and was simple to use, it allowed me to configure several various aspects of the bot, including what direction it is facing. The bots also fought well, unfortunately mine lost, but not before putting up a good fight. The process was easy for me to understand, but I am not sure if the same would apply to someone who is not as familiar with coding as I am, I think that any future progress should aim to make the process more simplistic.

Improvements and Observations

The main thing I got from the user feedback was that the interfaces I made worked very well, they found that the software simple for the most part and worked with little to no issues. The main part of the experience they reported back to me was that the games file structure could be difficult at some points to grasp and could confuse them. One other to note is that some users did say that the “bot types” option in the GUI was too vague and that they would have liked more detailing in the description of the bot types i.e. a full body spinner is a bot with a weapon that spins around the bot’s body 360 degrees.

I also noted that they would have liked to see the toolkit be able to handle the file structuring more i.e. taking their bots after they’ve been exported from the game and adding into the teams, or adding the actual adding of the output of the GUI into the bindings without needing to open the files themselves.

They’ve also noted that my own SuperAI “insanity\_2” behaved more aggressively than the other stock AI bots and that it at least seemed to have attempted to whip the rear blade in on rare occasion. This would be due to the bot having two sperate Smartzones and finding the back Smartzone closer to the bot it was targeting than the front blade. But this was seen to be a rare occurrence and the finding the bot more difficult to handle was in part down to the SuperAI, but part must also be placed in how well the bot was designed as well.

Overall, I think that if my toolkit did what I set out to do the start of this project, it made modding far easier to access for the users I asked to use it. Overall it was seen to be simple and the process was easily understood by all the users by then end of their first bot being setup to be AI controlled. The main issue they gave was the file structure, but I think this is important to learning how to mod a game.

In the aspect of the SuperAi I created people noticed a difference when the code wasn’t fully functional because of the OS issues when they used it. Before this issue arose, the backwards attack function worked without issue and was very effective at improving the stock games AI files. At the end of the project I would have to say that I cannot confidently say that the second and third objective were hit. I’m not sure if the issue would have appeared on an older system.

If more development could have taken place, then I would have added another option in the GUI it creates a very basic user (or called this new one basic and the current basic “intermediate”) to accommodate those users who found the file structure confusing. This version would deal with moving the bot that have been exported out of the game into the teams folder in the AI file and added the line of code made by the GUI into the bindings file for them.

On the SuperAI files I would have in future not bothered with these files and created a new version of the game in unity. This would have solved all the compiling errors that I had with the source code. Plus, this would have allowed me to create a game with something like steam workshop into the game. That and I could make the file structure more user friendly for teaching modding, and finally it would have made the debugging of code far easier meaning I could get a lot more done for the SuperAI files that I struggled with because of these issues with compiling and debugging.

Overall, I am happy with the GUI and the process I created for making user own AI bots to fight in game but I’m not 100% happy with self-created SuperAI files the one that did work was after a windows update stopped working, fully. But I think in terms of learning about code altering and project management I have made leaps in my ability to manage a workload. Also, I believe that in future if I do create my own AI for my own games in future I can do it without the need to have the engines default AI generator. And for creating simple user interfaces for less than advance users to interactive with complex systems, I think I am now very adept for the task but still could learn more.

# References

# References

(2012), I. (2017, November 15). *GET OVER HERE! Skyrim Mod Spotlight Ep. 9*. Retrieved from youtube: https://www.youtube.com/watch?v=kxGAbZHiurM

BeetleBros. (2018, aprill 18). *RA2 Download Database*. Retrieved from beetlebros.gametechmods: http://beetlebros.gametechmods.com/robotarena.htm

Bethesda Games Studios. (2018, april 10). *Creation Club*. Retrieved from Bethesda Games Studios: https://creationclub.bethesda.net/en

CodeGuru. (2012, january 30). *Storing String in a variable in a class file*. Retrieved from stackoverflow.com: https://stackoverflow.com/questions/9067303/storing-string-in-a-variable-in-a-class-file

Gabriel Interactive. (2017, November 10). *Robot Arena : Design & Destroy*. Retrieved from GabrielInteracive.com: https://garrielinteractive.com/#/865004006353

GameTechMods. (2017, November 10). *Complete Robot Arena 2 Mod Index. - WIP*. Retrieved from gametechmods.com: https://gametechmods.com/forums/modifications/complete-robot-arena-2-mod-index-wip/

GameTechMods. (2017, november 21). uncfl. USA.

R, L. (2001, november 10). *Online communities are chainging video games to make them btter, weirder, and much more wonderful*. Retrieved from ukBusinessinsider.com: http://uk.businessinder.com/video-game-modding-2015-7

# Ethical, Legal, Social and Professional Issues.

The main issue with modding a game comes the same as modifying any other code, copy righting of that software. The main reason why I didn’t create a modding kit for a game such as Skyrim or Fallout is that the games are still protected by copyright law.

However, the developers of these game Bethesda Game Studios are known to be big fans of modding video games, going as far as creating a service that allows players to pay for the best handpicked by the studios mods.

This is called the creation club and Bethesda describes it as (Bethesda Games Studios, 2018)” Creation Club is a collection of all-new content for both Fallout 4 and Skyrim. It features new items, abilities, and gameplay created by Bethesda Games Studios and outside development partners including the best community creators. Creation Club content is fully curated and compatible with the main game and official add-ons.”, This is the same with the majority of developers in the modern day, this allows this creativity of the player base whose members can code the ability to improve and add to their favourite games without worry of legal ramifications.

And because I am not profiting upon the work of the original developers, and that I am using this software for educational purposes then I fully believe I can use this content

# Appendix