Modding Toolkit For Robot Arena 2

To Do List (//delete before summation//)

It must make clear the problem being addressed and the objectives that the project aims to achieve.

* It must place the project in the context of the literature, or within the context of similar products.
* It must outline the approach taking to developing the proposed solution.
* It must outline the approach taken to perform verification or validation of the proposed solution.
* If appropriate, it should describe the tools used to support development.
* It should contain a critical review of what was achieved, rational of decisions made during development and evaluation of the project outcome.
* It must contain references to existing work.
* It must contain a description of Ethical, Legal, Social and Professional issues encountered during the undertaking of the project. You must enclose the Ethics checklist provided in Appendix II.
* A statement of originality (See Appendix I).

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 A.I bots using their own bots. I am doing this because the game itself is quite hard to edit without high knowledge of how the game is written. and majority of users have limited knowledge of how code in general works, yet alone how advanced Object Orientated(OO) code works.

Modding is described online as…

(1)“"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 (2) IAmPattyJack(2012)

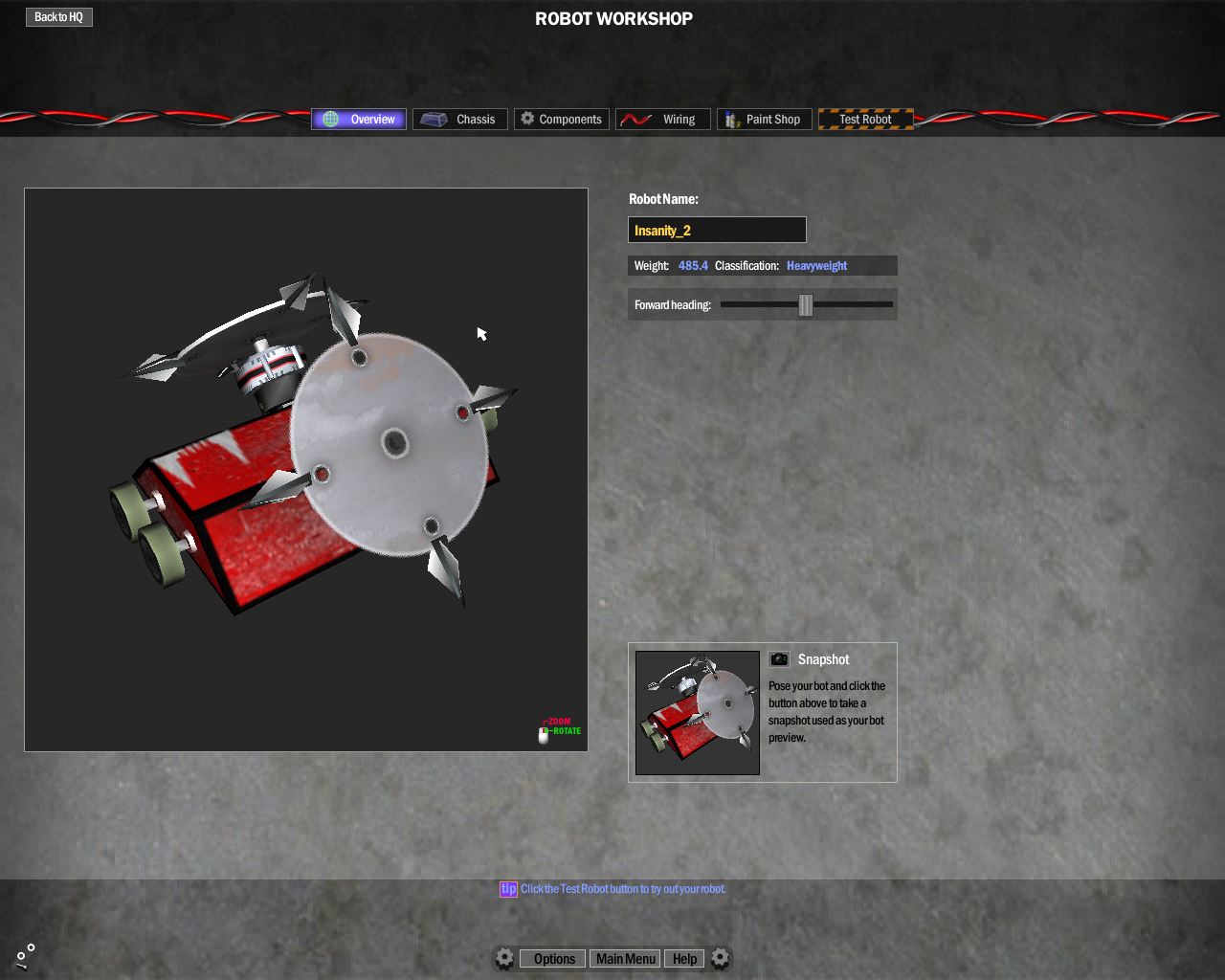
So, it’s very popular with people for use but not so much with creating them. This is as I stated earlier not many people know how to code, or how complex code works. This is required for modding, the main way of creating these now in the modern day is reading code and using a multiple number of custom 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 Gabriel Interactive.

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:

(3)“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 (4) <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 hope that I can get done by the end of this project and it is…

Being able to take a bot created as a user controlled robot and make it a A.I. controlled stock bot, using a GUI interface and removing all steps required in the folders of the game.

This should take up the bulk of my project, depending on how well it goes though I would also like to the able to do some of the following objectives…

* Add new components into the game that have been modelled in 3DS max.
* Add new textures to replace textures used in the game e.g. creating and mapping wheel skins.
* The ability for users to be able to add and remove arenas for the game.
* Giving the user the ability to customize the music in the game to suit their needs.
* The ability for the user to add and remove A.I. teams to the game and dictate the ones that show up in the main games rotation.\*
* And finally, the ability for the user to be able to customize the main games tournaments to suit how many bots fight in each round and the names/ descriptions.\*

As in the main objective of this project, all the above goals should also be within the GUI as one central modding application for the game.\*

And lastly, if this goes well for creating this games modding toolkit, I hope to be able to adapt the GUI interface I will develop for other title such as Skyrim, Fallout 4 and Age of Empires game series.

**The 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.

Once this has been done 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 it more efficient than the A.I. the stock game gives us.

After this we I can create new menus for users to create and add their own parts to the game using the same user interface style but in a different menu option.

After that’s been done then I may consider adding custom music, arenas, even menus styles.

The rest of the time will be the maintaining the application, by adding better A.i. options after tweaking stuff like accuracy.

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 ………………………………..

Registered Course of Study Computer Games Development

Date of Signing ……………………………….

Table of Contents

* Title Page
* Statement of Originality
* Table of Contents
* Background
* Methodology
* Results
* Critical Evaluation
* Conclusions and Future Work
* References

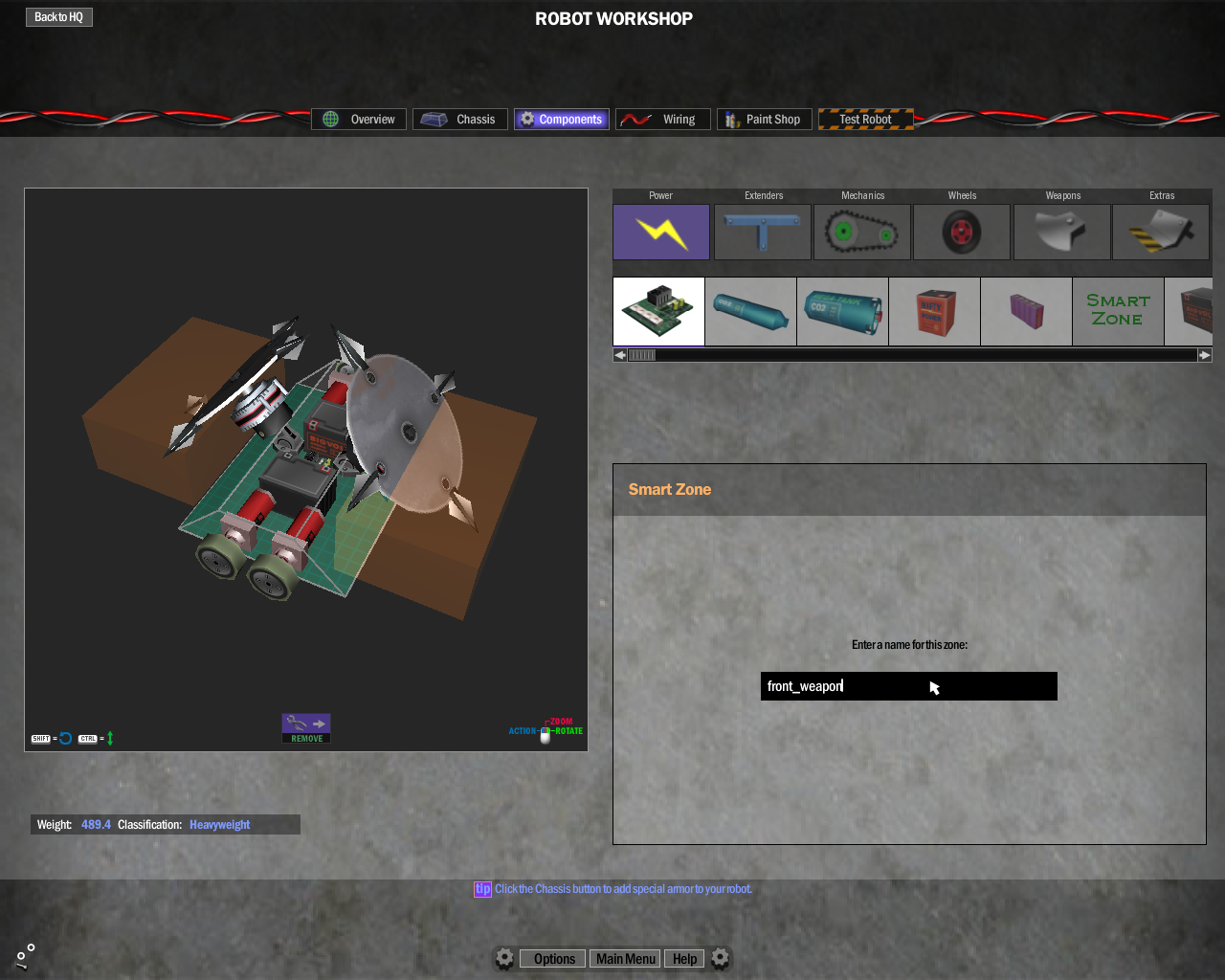
Background

* How the A.I. works,
* And how different types of A.I. are blinded to the bots.

So first thing, how does the A.I. know when to fire the weapon?

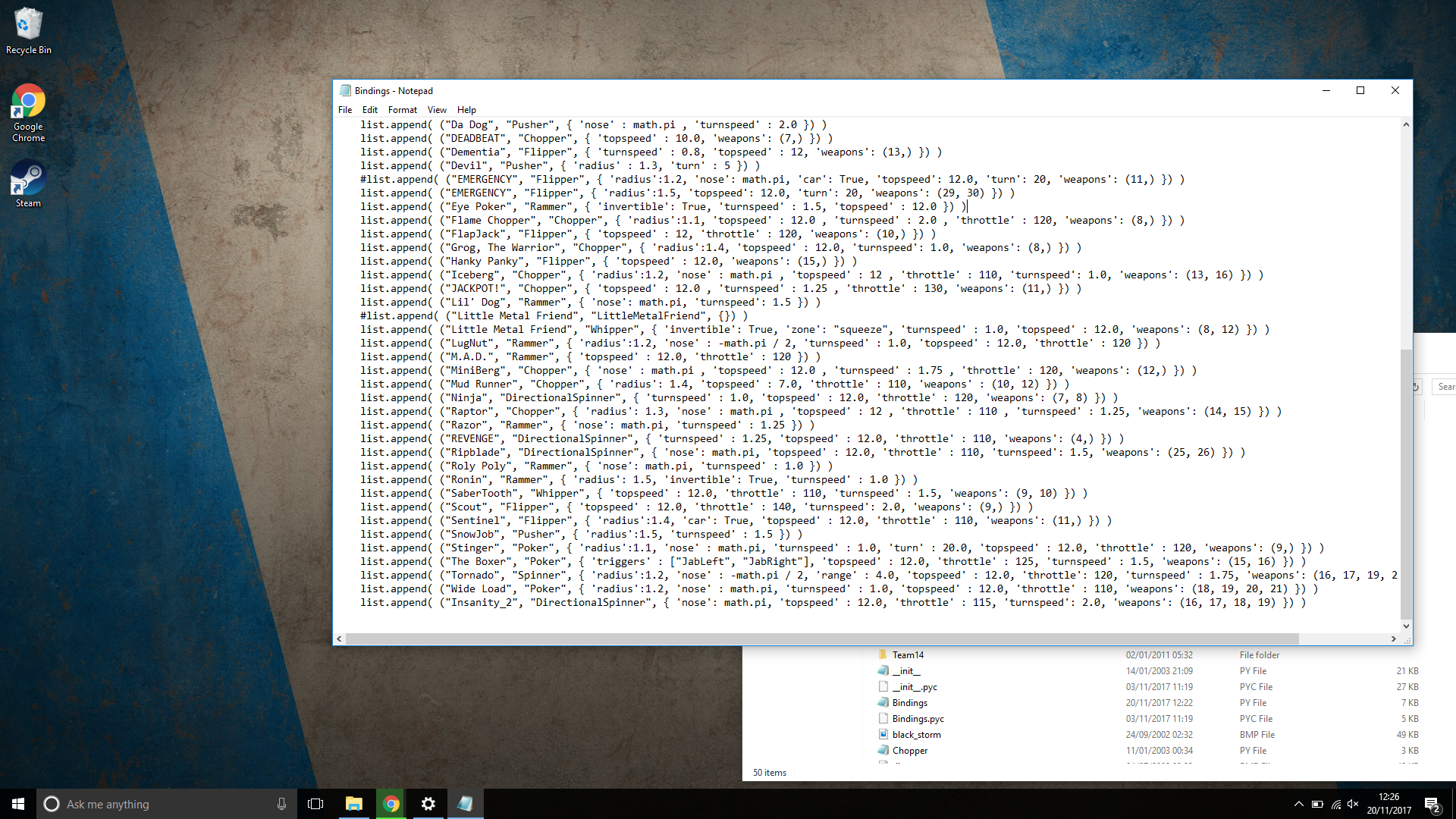
This is very simple, on every single bot within the game that uses A.I. uses one of 2 methods of knowing to fire the weapon. These are simple:

1. The simplest when another bot goes into the sight of the bot then it turns its weapon on. This method is revered for one type of bot, full body spinners, this is because a full body spinner can hit at any point around it, so if something is in the same area as your bot then you’re going to get a hit in.
2. If you have any other type of bot with an active weapon then you need to be able to know when a 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, in the bindings.py file (I will go into more later in this document), you can tell your bot the name of the smartzone on the bot. 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.

Outside of the way the .bot files and in game components, how does the A.I. get blinded to the bots in the game. Its done in the same way as other things in robot arena 2, a list that is read into a registry that is used in game to mash together all the maths files, bot design specific A.I. behaviour files and .bot files. This is done in a file 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”.

So how did I get my bot to work with the games in built A.I.?

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", "DirectionalSpinner", { '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 ai\_list.py 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 type of components have been used to create the weapon.

So the main things we need to do to create our own A.I. is…

Create a (weapon type)..py file,

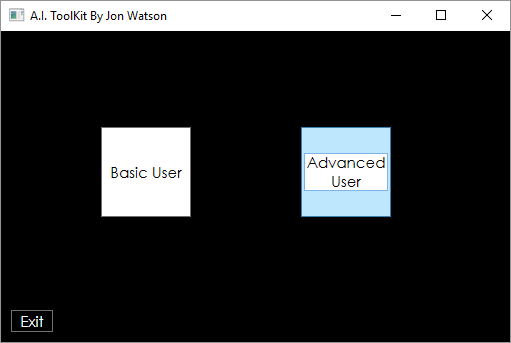
Add it to the bindings.py list.

Methodology

The first weeks of creating the project was spent creating the interface for the system, I’ve ended up making the user interface as a WPF application, so the main visual code is using XAML whilst the background logistical language is C#. 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’s 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 binded 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();

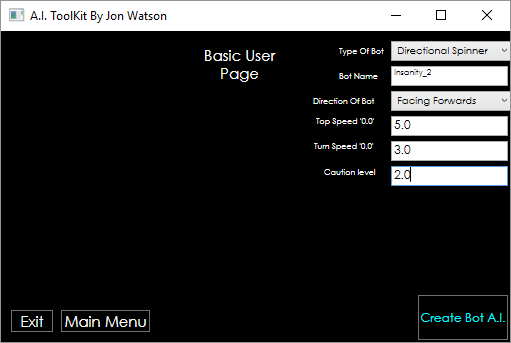
}

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,";

}

This could be improved 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

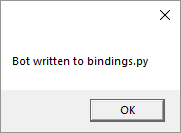
}

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).

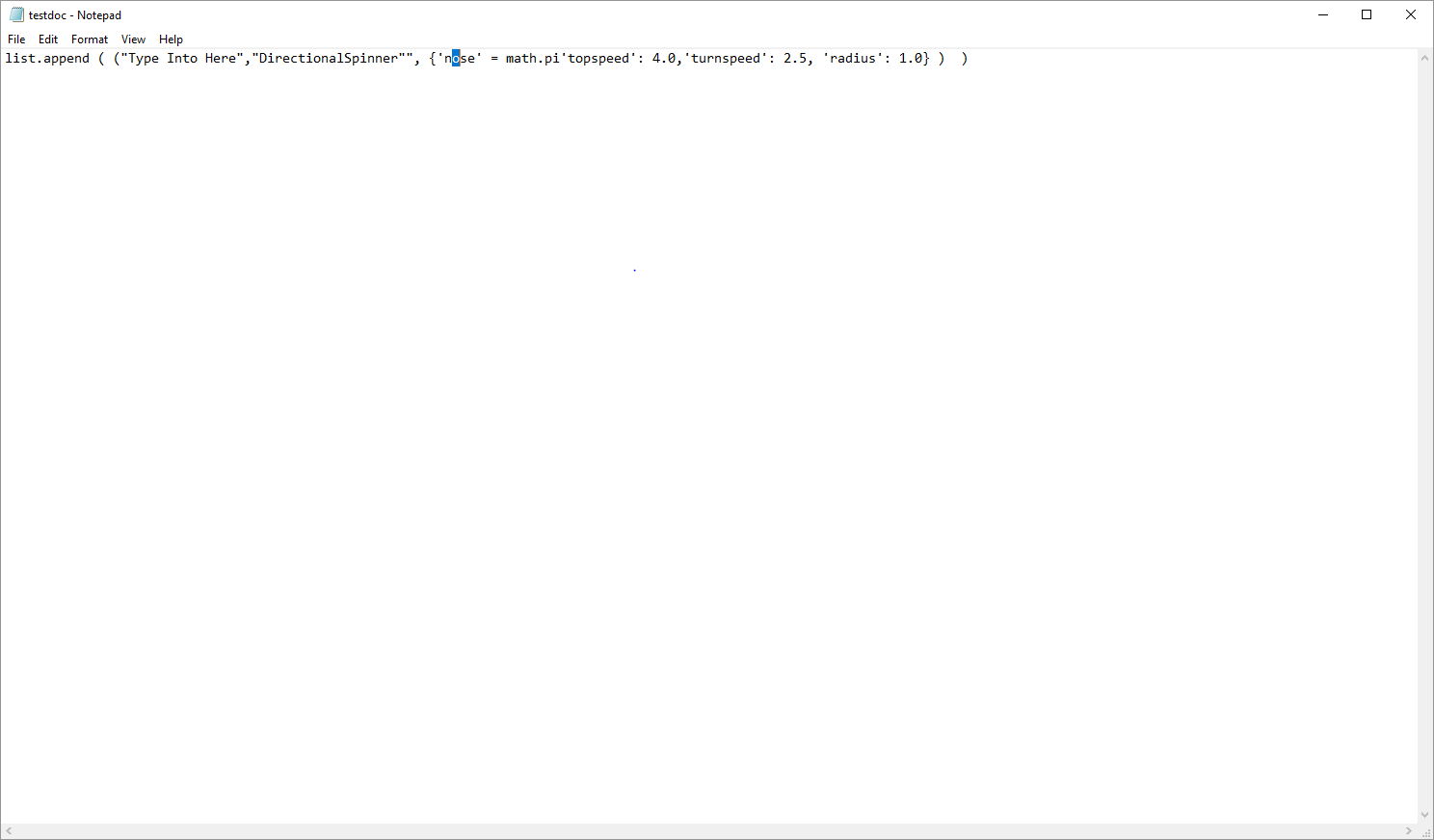
And finally, a window, telling the user the line has been done.[6]

[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



Results

Critical Evaluation

Conclusions and Future Work

References

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

* Letzter R (2001) *Online communities are changing video games to make them better, weirder, and much more wonderful.* Available at: <http://uk.businessinsider.com/video-game-modding-2015-7> (Accessed 10 November 2017 ).
* IAmPattyJack (2012) *GET OVER HERE! Skyrim Mod Spotlight Ep. 9.* Available at: <https://www.youtube.com/watch?v=kxGAbZHiurM> (accessed 10 November 2017).
* Robot Arena : Design & Destroy. [https://gabrielinteractive.com/#/865004006353/](https://gabrielinteractive.com/) (no date) (Accessed 10 November 2017).

Appendices

Ethical, Legal, Social and Professional Issues.