|  |  |
| --- | --- |
| Image result for game background art  Assignment title: ========  Component Code: Project 03 | Centre Name: Loreto college  Centre number: 32361  Candidate Name: Firaas Nasir  Candidate Number: 3271 |

Table of Contents

[Analysis 2](#_Toc32327506)

[Design 9](#_Toc32327505)

[User interface 20](#_Toc32327507)

# Analysis

* 1. **Problem Description**

Mainstream stealth games are targeted towards an older audience. Famous examples include the metal gear series, hitman and the batman arkham games. The problem is that due to these games including graphic content such as blood and gore, it means that children are unable to experience this genre which can easily be made suitable for them. It is evident that there is a gap in the market with very few children playing games of this genre due to the nature of them and the high age ratings.

In order to solve this issue and allow children access to this genre of games, I would like to make a retro inspired pixel game which is aimed at a younger audience that parents will give consent for them to play. My game will have significantly less violent connotations and won’t have a vulgar storyline; as a result parents should be able to consent to children playing this kind of game. Also, I will be using computational methods to provide a platform for the making of such a game.

Abstraction will be used to prevent the unnecessary use of hardware resources. Therefore the game will be a 2d rendition of a stealth shooter. The 3rd dimension can be removed due to the fact that the game is targeted at children therefore showing unnecessary movements in 3d may overcomplicate the game and could prove to be confusing for the consumer. Similarly, sprites will have simplified features so that they can be easily identified by the child and rather than trying to differentiate between sprites they can focus on gameplay. Furthermore the lack of a third dimension would result in less lines of code needing to be implemented therefore the strain on RAM will be reduced, as well as the processing power required being low. The file size of the game will also be small as a result.

The game can be broken down using decomposition into smaller chunks and tackled individually. Individual aspects of the game can be broken down into classes so that object oriented programming can be implemented. For example the controls for computer controlled sprites can be specified in a single class and if more sprites are required they can be instantiated. This will allow for better organisation of the code therefore it will be easier for the programmer to understand what is going on in the code after taking a long break.

Branching and loops will also be used in the game to make it more appealing and more entertaining to play. The logical operators will allow there to be multiple different outcomes that will be created in the game rather than it following a linear structure, this factor of uncertainty of what the outcome will be (perhaps to live or die as a result of an event) creates a drive for the user to play the game and makes it overall more fun. If/else statements can be used to determine if the user is deciding to move their sprite by confirming key presses e.g. arrow key presses can correspond to movement in game. The timer subroutine will be looped in order to update the game constantly and refresh activity on the screen so the user can interact with the interface.

* 1. **Stakeholders**

This game is designed to be popular with a young demographic, mainly children under the age of 12 who are unable to experience the action/ violence of stealth shooters designed for older audiences. Therefore the stakeholders are generally going to be young children with little gaming experience.

Children tend to play flash games on computers due to their lack of complexity and small learning curves. An example would include ‘boxhead’, where the player takes control of a sprite from a top down view and tries to kill as many enemies as possible using only keyboard presses. The stakeholder for such a gaming demographic is Aashir Nasir who is well acquainted with simple flash games. He is also under 12 years of age which means that he is unable to play games of stealth shooter genres therefore he will benefit from a game of that genre becoming child friendly.

* 1. **Research the problem**

My question for Aashir Nasir, who represents the young gaming demographic:

1. Have you ever played a stealth shooter before?
2. Are you allowed to play games with violence in them?
3. If not, would you like to experience a similar genre?
4. Do you prefer controls that take advantage of the mouse, keyboard or both?
5. What features do you think stand out in a stealth shooter that defines it as a genre?
6. Do you have any further comments to add?

Question 1 provides some insight into their experience with the genre of stealth shooters. This lets us know if they have any prior knowledge regarding this genre, if so it makes their assessment more credible.

Question 2 and 3 shows whether they have played any violent games and inform us to whether they want this problem to be solved.

Question 4 will shed some light on how the controls of the game should be programmed so that they can be used by the target audience efficiently and comfortably.

Question 5 and 6 asks them to highlight any distinct features of a game in the specific genre so that when programming it these features should not be left out. This is so that the game can provide the most authentic experience possible.

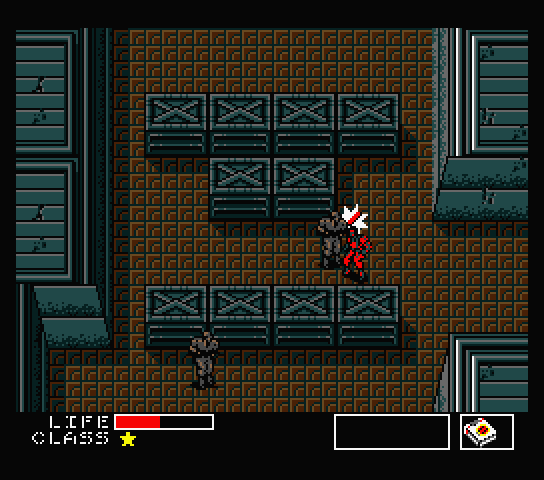
**Analysis**

The interviewee had never played a stealth shooter however he can confirm that he has played shooting games before. He confirms that he isn’t allowed to play violent games and he wishes that there was a way to enjoy the same content with toned down graphic scenes.

He also stated that he would prefer controls that made use of both keyboard and mouse functions because “it feels right”. Furthermore in terms of the genre he pointed out that a stealth shooter should have ways of taking out enemies silently and not so silently, therefore if the protagonist is spotted there should be more efficient means of taking out hoards of enemies once found out; there should also be vantage points for the player to hide and observe the enemy without being spotted, such as boxes; typically enemies should do rounds and should only chase the player when they fall into the enemies vision; close quarters combat should also be coupled with ranged attacks to produce an authentic stealth shooter game.

**Current solutions**

‘Metal Gear 2: Solid Snake’ is an action adventure stealth game which is viewed from the top-down point of view. The player takes control of Solid Snake who must rescue Dr Marv. Enemies will do rounds and follow the paths that they have been set and look around to see if there’s any trouble, the player must try and stay out of their vision to avoid being spotted. If he is spotted, enemies will engage the player and find their way to him using a path finding algorithm and a head on fight ensues. The game accommodates saving progress as well as scores so the player can take breaks. The game compels the player to continue playing so that they can advance in the story and find out what is happening next; the story is conveyed through simple text dialogue boxes.



‘Ronin’ is a far more complex example of a stealth shooter as it employs mechanics such as slow motion and the use of a skill tree which allows you to unlock new abilities. In this game you play as a ninja who infiltrates establishments crawling with guards, you must take out the guards and stealth is the most efficient way to do so. You can crawl on walls and use a grappling hook to your advantage so you don’t end up being spotted. Unlike metal gear, this game makes use of both a keyboard and mouse. Also, unlike metal gear it isn’t played from the top down perspective; it is played like a 2d platformer. Although it has a story, it isn’t the main focus of the game, rather it is driven more by its great gameplay.



After analysing the games I can deduce what makes an action stealth game. Although ‘Metal Gear 2’ has a compelling story, to some it may get monotonous trying to read all of the dialogue to extract the lore. ‘Ronin ‘shines as it doesn’t need a story to convince the player to come back rather it’s well designed gameplay is enough to suffice. Therefore since I don’t want to overwhelm my target audience of children with a complex story I’d rather take the approach that is demonstrated by ‘Ronin’ which is to create challenging levels that the player wants to beat. Furthermore both games employ a difficulty setting, in metal gear this means an enemy may be able to spot you but not be alerted a certain number of times; if it’s set to easy this number would go up. Ronin has a range of enemies too with different abilities and can be seen as mini bosses, this also makes the gameplay a lot more interesting as the enemy types vary.

**Key Features of the solution**

**Path finding**

Once the enemy has spotted the player they will go into alert mode. This means that rather than following their previously designated routes, they will instead use a path finding algorithm to locate the player and follow him.

**Basic movement/combat**

The player will move with the ‘W,A,S,D’ keys and use the mouse to point at a target before deciding to shoot by clicking, the bullet will travel to the last position of the mouse when it was clicked. Furthermore, the player will be able to switch weapons with the mouse wheel. There will also be a silent close quarter’s combat weapon that will be included to ensure the enemy won’t be alerted when the player attempts a kill.

**Objectives/Score**

Main objectives will predominantly include the player having to reach a certain extraction point. When the point is reached he will be presented with a screen showing the score. The score will be made up of how many enemies you kill and how fast you complete the stage. These scores will be stored in a text file and a high score system can be implemented.

**Inventory**

There will be an inventory displayed at the top of the screen. It will depict what weapon the player has and the number of bullets. The inventory will also come with a health bar which will show how much health the player has before he dies; it will go down when the player takes damage from enemies.

**Hiding spots**

The player will be able to enter these hiding spots which will be blind to the enemies’ vision, this allows the player to wait for enemies to pass by or take a breather.

**Pause screen**

The player will be able to rest while the pause screen is up and will be given the option to save the game and come back to it later.

**Limitations of my proposed solution**

The main limitation is that I am not highly experienced with programming. This means that when t comes to implementing complex algorithms such as a path finding algorithm, I will have to rely on the internet for tutorials to guide me in the right direction in order to produce the results that I want. The problem is that with vb.net, the tutorials are for quite basic problems and the information on complex algorithms is scarce and of low quality therefore it may take longer to learn/ implement systems such as collision detection. This lack of information and experience means that it may take me longer to reach my deadline. To solve this I must ensure that my code is completed in a modular rather than linear way.

Furthermore, the short deadline for the project (April), may limit the outcome of my project. For example, since one of the main objectives of my project is to make an entertaining game I will have to make an assortment of interesting and complex stages, however due to the nature of this A level project time is limited and therefore it is unreasonable to think that I could make enough stages to keep the game at a length that players will deem acceptable. Thus I will have to limit the number of levels and demonstrate what this game could be if it was allocated more stages in order to attract the attention of my target market.

Another limitation is that I am inexperienced with graphics editor programs such as adobe illustrator. This means that the graphical appeal of my game may be limited to very basic art that may not accommodate pretty backgrounds and scenery that is usually found in most professional games.

I must also mention that since I am programming my game on visual studio in a language created by Microsoft, my game won’t be portable. This means that I will not be able to easily transfer my game to other operating systems that consumers may want to play on. Therefore the nature of the programming language limits that playability of the game to windows only.

**Hardware and software requirements**

Since my game will be compiled and executed on visual studio, I have listed the requirements that are needed to run visual studio. As a result, meeting these requirements will optimise the overall experience with my game.

|  |  |
| --- | --- |
| **Requirements** | **Justification** |
| Software |  |
| Operating system: Windows 10 version 1703 or higher/Windows 8.1/Windows 7 SP1. **(64 bit is recommended)** | Visual Studio 2019 will install and run on the following operating systems. Without them, you won’t be able to run the game. |
| Hardware |  |
| Processor: 1.8 GHz + | The processor will need to quickly perform functions such as executing tasks and drawing graphics; it would cause the game to appear slow if a poor processor was used. |
| RAM: min 2GB, recommended 8GB |  |
| Hard disk: 1gb+ free space | 800mb is needed to download visual studio. Furthermore, my project shouldn’t take up more than 200mb. |
| Graphics: any video card that supports a minimum display resolution of 720p | Minimum requirement to render the graphics of the game. |
| Keyboard | This will be required to perform actions in the game; such as moving. |
| Mouse | This will also be needed to perform actions in the game, such as navigating the menu. |
| Monitor | This will display what is going on in the game to the user, so they can react accordingly to what’s happening on the screen. |

**Success criteria**

|  |  |  |
| --- | --- | --- |
| **Requirements** | **How to test this** | **Justification** |
| The game should have a pause function . | Check by pressing the button ‘p’ which will cause a pause screen to appear in front of the player, giving them options to save the game/quit. | A pause function will allow the user to take breaks by giving them an option to save progress and come back to it later, it will also allow the player to tweak settings while playing the game. |
| Crash prevention – the game shouldn’t crash while the player is playing the game. | Enter erroneous inputs where possible, they shouldn’t lead the game to crash. | Game crashes can lead to players losing their progress which is not ideal as they would have wasted their time. |
| Have a save file - data about the player’s current status can be exported to a text file that can be later read when loading a save. Things such as location score and inventory will be recorded. | Check the text file after the save has been made to ensure all of the data has been exported properly. | A save function is needed so that the player can later pick up from where they left off by saving their progress to a specified location. |
| Save files should be loaded. | A text file will be read and the game will load the information on the screen accordingly e.g. last position and what was previously in the inventory. | It will allow the save files to be loaded and the player can resume from where they left off. |
| The weapons will be fired in relation to the position of the cursor - the position of the mouse must be constantly recorded and a crosshair will follow. When the player decides to fire, the attack will begin its course to the last place the cursor was when it decided to fire. | Check by shooting a bullet and seeing if it reaches the cursor. | This will allow the player a visual representation of where their attacks are going and as a result, they can aim their attacks in order to kill enemies. The mouse is an effective way for the player to point out where they want to attack as swiftly as possible. |
| ‘w’, ’a’, ’s’, ’d’ will control the players movements. | ‘W’, ’A’, ’S’, ’D’ will cause the players sprite to move ‘up’, ’down’, ’left’, ’right’ respectively. | The player will be able to confortable use the mouse while moving the player as the directional controls will be on the far left side of the keyboard whereas the mouse is typically on the right side. It is ergonomic and generally, the player should be most comfortable while playing in this position. |
| Enemies will have ‘vision lines’ - unlike 3d vision cones, this is a 2d game so lines will be used. If a player enters the enemies’ vision line they should be alerted of the players position and the path finding subroutine should be executed. | The player should move past the enemies’ un hindered line of sight and the enemy should detect it. | This gives the game an element of stealth as it objectively pushes the player to avoid the enemy line of sight. It has to detect the player in order to pursue them. |
| Enemies should track the player - when the enemy is alerted of the players whereabouts (from their vision), they should apply a path finding algorithm to approach the player. They should be able to find the best way to reach the player given their current positions. | After the ‘vision’ (the player has been spotted) has been confirmed, the path finding function should calculate the shortest distance and pursue the player. | This will give a more realistic feel to the game if the enemy can track the player once alerted. |
| Clear user interface: display items/stats - the game should show remaining ammo and health at the top of the screen, as it is used up the number should decrease accordingly. The weapons in use should also be displayed in a clear fashion. | This can be measured post testing when the stakeholders have had a go at the game. They can fill out a questionnaire on how clear the UI was. The questionnaire will be of quantitative values as it is the most efficient way to measure results. | This gives the player information and lets them know if they should save their bullets/health and whether they should play conservatively. Without a clear UI, players may have a hard time navigating the game or just playing it in general. |
| Collision - the player should collide with bullets/walls and an action should be carried out accordingly. | All four cardinal points of the picture boxes will be tested against all of the cardinal points of the other picture boxes that may have a chance of colliding with the player. | Perfect collision ensures that the player doesn’t fall out of bounds. It allows the game to check if the player is interacting with the environment e.g. being shot and will act accordingly e.g. take damage. |
| Must be child friendly. | Have the stakeholder fill out a questionnaire on whether the game brought about any violent tendencies or caused the user discomfort. They can rate how comfortable they felt playing the game on a scale of 1 to 10. | As this game prioritises the wellbeing of children, it is essential that the stakeholder deems it worthy of being played by children. |
| Variety of weapons should be fun to use. | Have the stakeholder try all of the weapons and see if they found themselves switching between the weapons a lot. | Frequently switching between weapons indicates that they are each unique in their abilities and play style. The added layer of just switching between guns mid game would make the game more interesting. |

# Design

**Decomposition of the problem**

|  |  |  |
| --- | --- | --- |
| **Decomposed solution** | **Why I decomposed the solution** | **A brief overview of the solution** |
| Player | The player class is very important as it is integral to the game and it will be seen by the user all the time. I have decided to decompose what makes up the player class into its algorithms such as movement, collision detection and its properties. | I must clearly separate the features of the player class so that they can be efficiently tested. Algorithms will be kept within separate subroutines such as the controls for movement, collision detection, interaction with objects and the change in properties as the game progresses. |
| Enemies | The enemies will be plentiful in numbers and that’s why this class must be decomposed to ensure that they are coded in the most efficient way possible. | The enemies will also incorporate methods similar to the player such as collision detection, however the path finding algorithm will be the main focus. I am going to efficiently incorporate this process into the class so that it allows for a good gameplay experience. |
| Collision detection | Most objects will be making use of the collision detection system, therefore by decomposition I will be able to reduce the duplication of the code and therefore make it more efficient as a result. | The collision detection system wills most likely need to be a function as it will survey whether the object is colliding and hence return a true or false value. This will let the object decide what to do when a collision is detected. |
| Player movement | The movement subroutine is important as it will be constantly checked while the game is running. The player sprite won’t be able to move if this subroutine isn’t made. | This subroutine will work as a method in the player class and will be called in the update subroutine constantly. It will allow the player to control the movements of the main sprite with arrow keys. |
| Weapons | The weapons will be used as a tool for the player to progress further in the game. It should be decomposed because it has many different components such as weapon variations, shooting and bullet travel | If firearms are selected, the player must be able to shoot wherever they want accurately; if the player is to select a melee weapon then they too should be able to attack in the direction they please. The difference will be that a bullet will be spawned for a firearm. |
| Saving | This algorithm is important because it will allow the player to save where they left off and resume game play whenever they like, this allows the player to take as many breaks as they like and they don’t risk losing their data. | Once the player decides to save the game, the save subroutine will be executed. The save subroutine will store essential data such as health, player position and inventory in a csv file so that it can be read from later and loaded back into the game. |

*Why am I using this approach to help decompose my problem?*

I am using a top down approach because it allows me to think procedurally and essentially break down the complex task of designing a game into smaller chunks of problems which are easier to manage. I have used stepwise refinement to help me logically move through the levels of abstraction, going from high to low levels and refining the preceding algorithm with each stage. To complete sections of the whole code, I have identified sub tasks which give me a clear idea on what algorithms I need to design to complete each task and use computational thinking skills “thinking ahead” to determine the inputs and outputs and “thinking logically” to see where decisions may be required and what effects it may have on the wider solution. The top down approach has helped me to organise the order in which I will look at programming the overall solution. The process of decomposing my solution in such a way will prevent me from arriving at hurdles like contemplating what algorithm should be programmed next; instead I will have a diagram to refer to and be able to follow a clear plan.

***Computer controlled sprites:***

I have identified the subtasks that need to be developed in order to complete the computer controlled sprites part.

**Designated walk paths:** Enemies that stay idle will be uninteresting so to add depth to the game, they will walk along their designated walk paths to patrol the area. This will increase the likelihood of the the player being detected by the enemies’ proximity sensor. When following the walk path they will be at normal speed but when the player is spotted, the enemy will start the path finding algorithm and chase the player at an increased speed.

**Health counter:** The enemies will need a health counter which will change its value according to certain factors. When the enemy comes into contact weapons such as bullets, their health should decrease by a certain amount. When the health value of an enemy reaches 0 or below, they should be removed from gameplay as to not hinder the player.

**Collision detection and response:** When an enemy is colliding with any picture box, the collision detection algorithm should return the value ‘true’. Once the algorithm has confirmed that the enemy is colliding with an object it should decide what to do next; when colliding with walls, the enemy shouldn’t be able to pass through and when colliding with weapons, the enemies’ health should decrease since that means the enemy has been hit.

**Path finding algorithm:** When the enemy has spotted the player with its ‘proximity sensor’, the path finding algorithm should begin. Once the most efficient path to the player has been identified, the enemy must take that route in order to reach them.

**Proximity sensor:** The proximity sensor algorithm is there to essentially provide the enemy with vision while it is patrolling. This algorithm should act as the enemies’ line of sight and return the value ‘true’ if the player enters the specified region. If the player happens to enter the line of sight, the path finding algorithm should begin and the player will be chased.

**Iterative development approach**

I will be using an iterative approach to developing a solution by completing one area at a time:

* Stage 1: Make home screen
* Stage 2: Player movement and controls
* Stage 3: Player health and score counter

Version 1

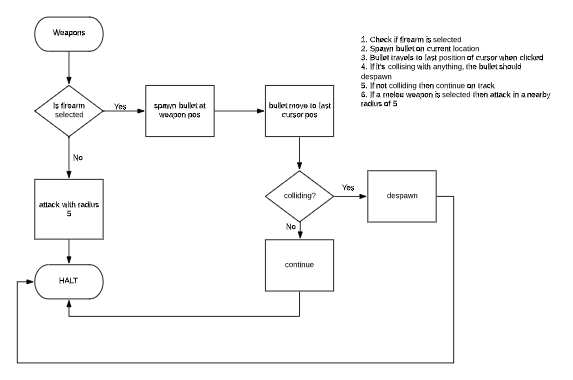
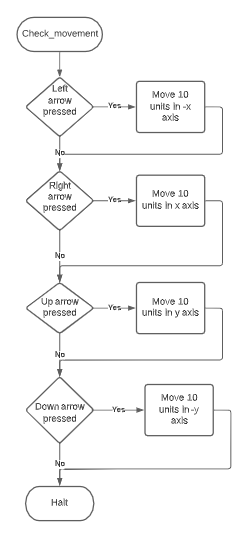
* Stage 4: Computer controlled sprites walk paths
* Stage 5: Computer controlled sprites Health counter
* Stage 6: Make walls which are able to be referred to as tags
* Stage 7: Weapons/ bullets with inventory
* Stage 8: Player collision detection and response

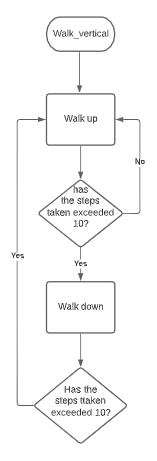
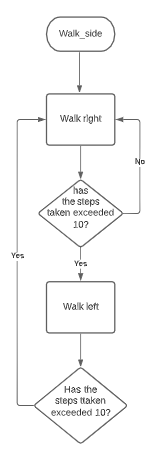
Version 2

* Stage 9: Computer controlled sprites collision detection and response
* Stage 10: Computer controlled sprite proximity sensor
* Stage 11: Path finding algorithm
* Stage 12: Camera that locks onto player

Version 3

* Stage 13: Additional map design for new level
* Stage 14: Login screen with score saving features
* Stage 14: load/save feature

I have chosen to complete the basic algorithms in version one such as movement for the player and computer controlled sprites. This is because I want to have a basic understanding of what my key classes and variables will be when I proceed to the later versions of development. The version one prototype will serve as a base to be built upon by later version with more complex algorithms. Version two is where I have included most of my complex algorithms such as collision handling and the path finding algorithm. The development of these complex algorithms in version two rely on the completion of version one therefore I have chosen to do them after. I have decided to keep the tasks in version three basic as it will likely be developed at a busy time, furthermore it will give me extra time to go back and try to make my code more efficient. 



**Proposed screen designs and usability features**



The player will select the options from the menu by left clicking with his mouse; this is more convenient than using a keyboard because moving a mouse is a swifter action than using the arrow keys to select options.

The order that I have chosen for the buttons is depicted in the picture, this is for the convenience of the player as the order reflects what they are likely to click first.

The title of the game is clearly stated at the top so that it is the first thing the player reads

This will be the screen that the player will be greeted with when they first run the game. I have gone with bright colours that stand out to my younger audience of players so that it will give them a pleasant feeling when loading the game, furthermore it will be easier for them to see bright colours rather than dark ones. Also, I have chosen to use a large font size and one word for the buttons so that the player needn’t strain themselves when deciding what to click.

**Key variables and structures**

This record structure will store data about the player regarding the information needed for a game to be saved; it will be written to the ‘Save’ database. They will be stored ass a csv file so that they can be updated and read from at any time.

|  |  |  |
| --- | --- | --- |
| **Field name** | **Data type** | **Description** |
| Health | Text | This will store the player’s current health. The health must be remembered so that its change doesn’t handicap the player upon loading the game again. Therefore I have decided to store it as a record. |
| Position | Text | The position will be stored to determine where the player last left off before they saved the game. Upon loading, the game will spawn the player where they were last present. |
| Ammo | Text | The number of bullets that the player last had will be stored so that the player can resume the game with the same number of bullets that he was previously playing with. This allows for fair gameplay as the player won’t be put at an advantage or disadvantage for saving the game. |
| Inventory | Text | This will store what weapons the player had so that upon loading, the player can continue to play with the same items. |
| Score | Text | The Score must be stored so it isn’t lost during a save. This means that the player won’t lose points when taking breaks and they will have an equal chance of setting a high score to people who take fewer breaks. |

Key variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Subroutine** | **Purpose& justification** |
| PlayerPosX, PlayerPosY | Single |  | These variables will be the x and y vectors for the player and will track where the player is in the game at all times. |
| PlayerColliding | Boolean |  | This will determine whether the player is colliding and will be useful in selection. It will be useful in determining whether the player is being shot or walking into a wall. |
| Score | Integer |  | The score variable will be updated when a player accumulates points, it will increase accordingly and display how many the points the player has in total. |
| Health | Integer |  | This will determine how much health the player is on, its number will go down when damage is taken and when it reaches 0, the appropriate subroutine will be called. |

Validation:

The only part of my program that requires possible erroneous human input is the part where the player enters their name for the high score being recorded. Therefore I have decided to validate this entry before it is stored.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input value** | **Method** | **Description** | **Justification** |
| UserName | Length Check | The name must be a certain length so that when the scores are being displayed, the name mustn’t stretch so long as to go out of the screen or overlap other text. | A length check can be used to verify if the user has stayed within the character limit. This will ensure that overly long names won’t be entered in this section. |
| UserName | Presence check | This will ensure that the user doesn’t leave this section blank, it won’t move forward until there is something in the input. | The presence check will ensure that the user doesn’t leave the input blank and use it as a name. This will ensure that the person getting the score is given an identity. |

**Test data for development**

I will be testing my program as it is being developed so that I can minimise troubleshooting at the end, I am following a black box approach to testing the final beta. These are the values that I will be testing to see if the algorithm that I programmed is a success.

I am going to test the collision between bullets and sprites in relation to their ‘hitboxes’ with the following test data:

|  |  |
| --- | --- |
| **Test data** | **Type** |
| Bullet in > 5 radius of enemy leads to collision | Valid |
| Bullet in <5 radius of enemy leads to collision | Invalid |
| Bullet in =5 radius of enemy leads to collision | Invalid |

This test data should accommodate all of the outcomes in relation to the distance of the bullets colliding. The distance will be checked with an on screen label which will confirm how far the bullets are from the enemy; this will allow the test data to be easily checked.

|  |  |  |
| --- | --- | --- |
| **What is the test for?** | **Test data** | **Justification of test data** |
| Enemies should spawn at their designated points | Variable that stores the X/Y co-ordinates of each enemy. | The vectors must correlate with the positions on the game screen; the enemies must spawn in locations that seem normal. |
| Test if enemies detect player | The player will walk into the enemies ‘line of sight’ and if so, the detection function should return true. | The player must trigger the enemies’ detection subroutine when they walk into the line of sight. |
| Path finding algorithm determines the most efficient path | The enemies’ time to reach a certain point in comparison to a human controlled test enemy. | To see if the enemy has chosen the most efficient path, ill first time how long it takes for the enemy to reach a certain location in a test level. Then, the human controlled sprite with the same movement capabilities will attempt the same course. The results should be similar to assume that the enemy has found the most efficient way to travel. |

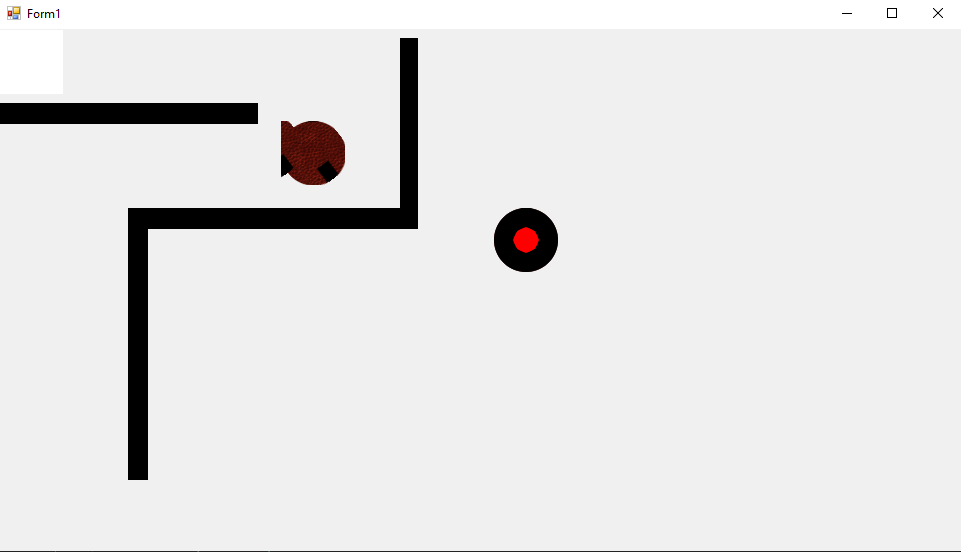
Menu:

|  |  |  |
| --- | --- | --- |
| **What is the test for?** | **Test data** | **Justification of test data** |
| Test if the user has entered the correct number of characters for a username | aaaaaaaaaaaaaaa | The invalid test data being used is going over the character limit; the algorithm should reject this input and ask the user to re-enter. This will allow me to test if this validation algorithm has worked. |
| Test if the user has entered anything in the username box | \*Will be left blank | Invalid test data will allow me to test if the presence check works. |

**Development and testing**

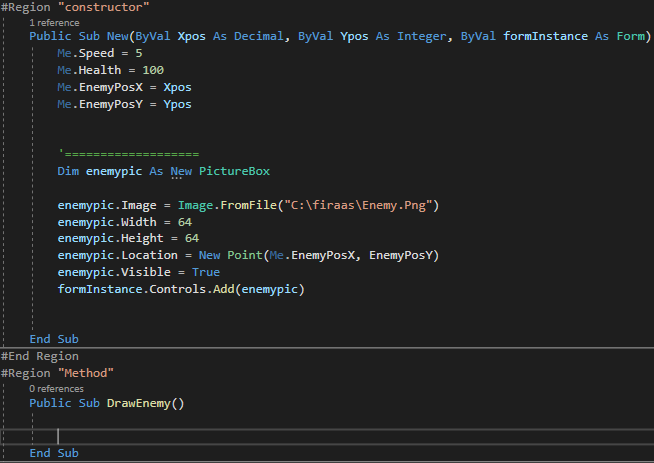
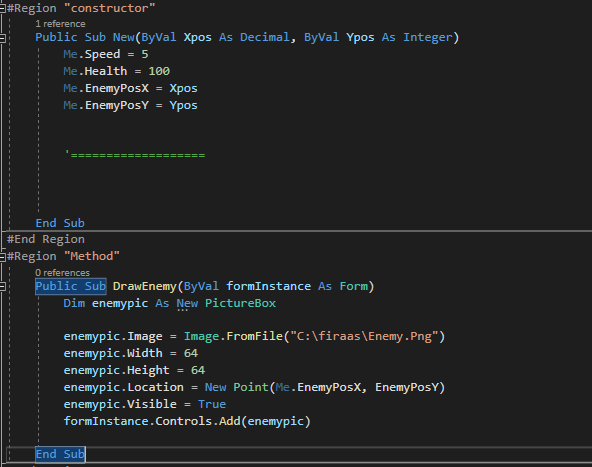
**Version 1**

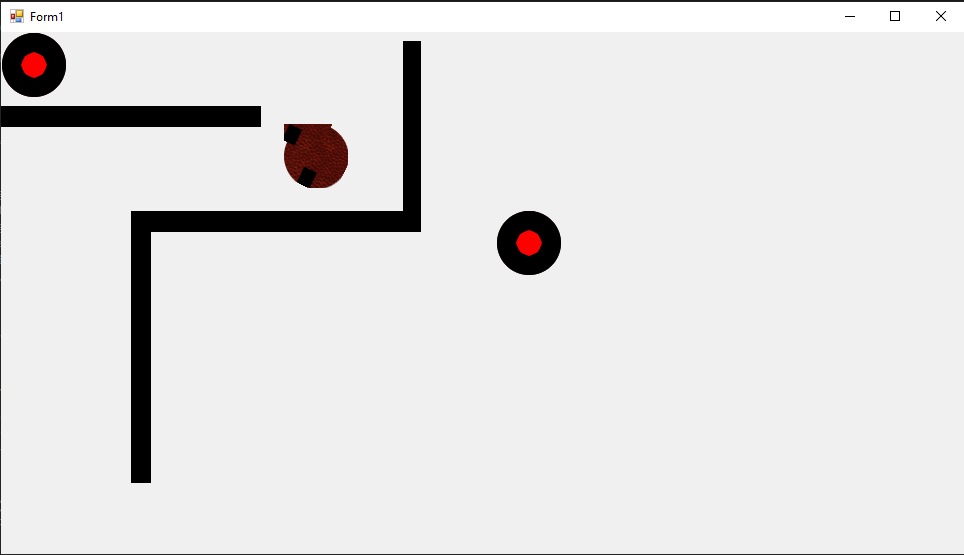
***Instantiating enemy picture box***



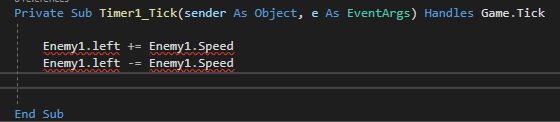
I created an enemy class with attributes, constructors and methods which would be ready for instantiation and creating many enemies. I created attributes which were the main variables referred to when using an enemy object, I used the constructor to set these values to their designated pre-sets and finally I made a method that would make the picture box for the enemy. As depicted by the arrow and circle, upon instantiation my enemy picture box would remain blank.

To remedy this, I moved the code that was creating the picture box out of the method and into the constructor since it was something that all objects of the class required.

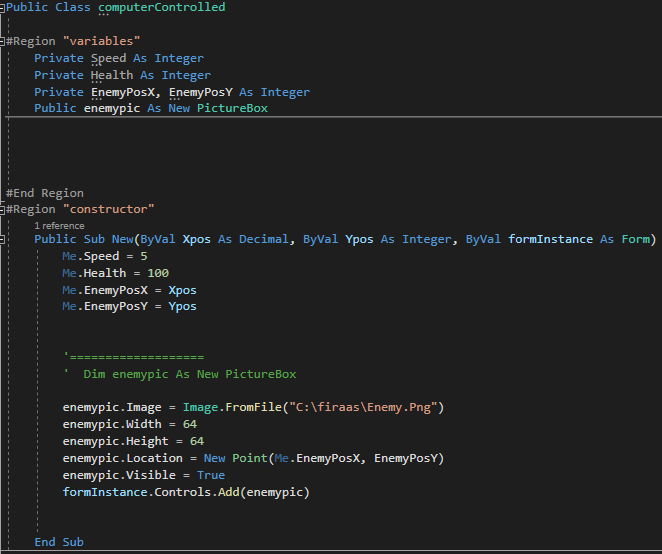




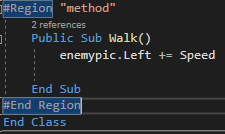
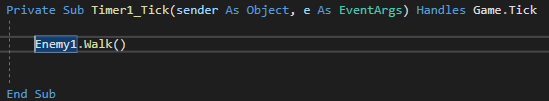
***Enemy movement***



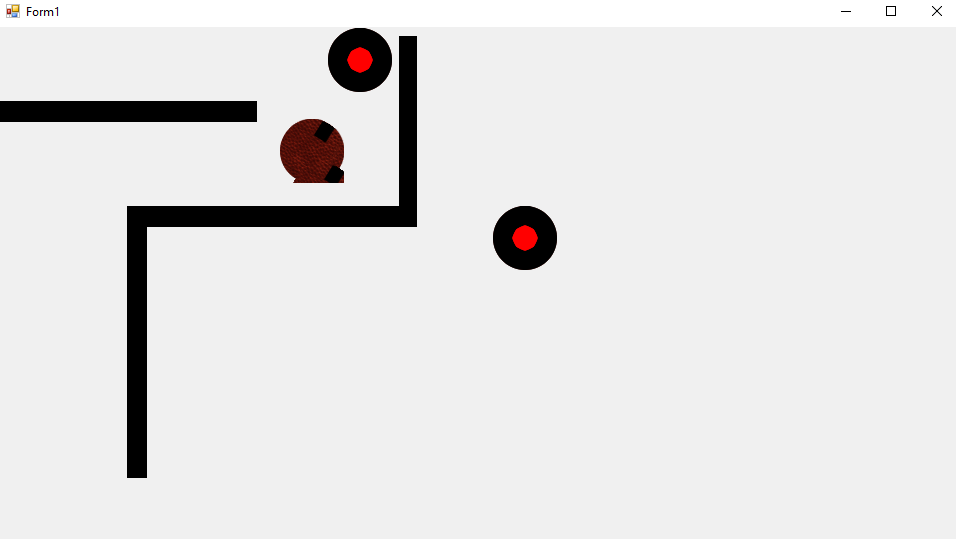
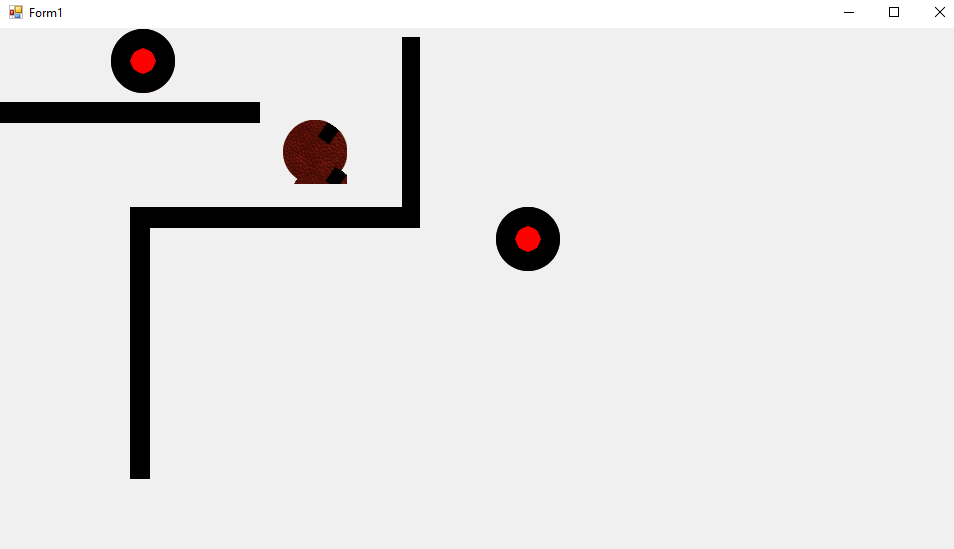
I tried to get the instantiated enemy picture box to move left and right however it wasn’t referring to the picture box, rather it was referring to the class (‘computerControlled) as a whole which didn’t let me manipulate the picture box. I realised that I wasn’t able to refer the picture box because it was a private variable, inaccessible from the main solution. To remedy this, I made it a public variable and declared it among the rest of the variables in the ‘computerControlled ‘class.



I then realised that despite being a public variable, I wouldn’t be able to manipulate the picture box (‘enemypic’) outside of the class. Therefore I realised that I needed to make a method that will control movement in the ‘computerControlled’ class and it would be called through a timer in the main code.



**REMEDIAL ACTION**



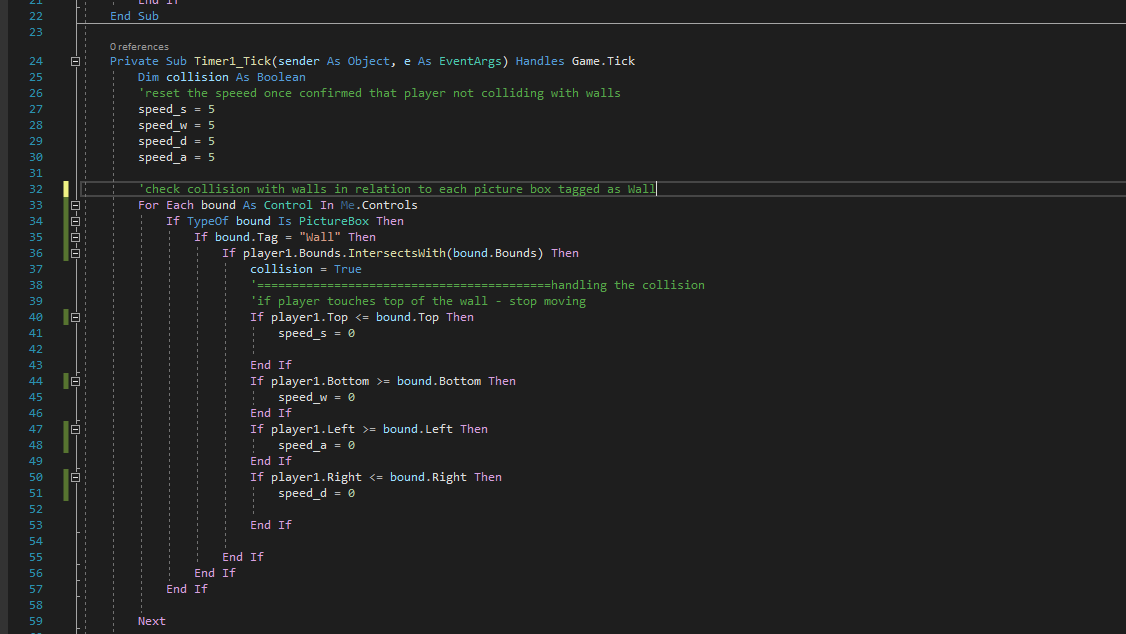
TEST

***Version 2***

***Collision***

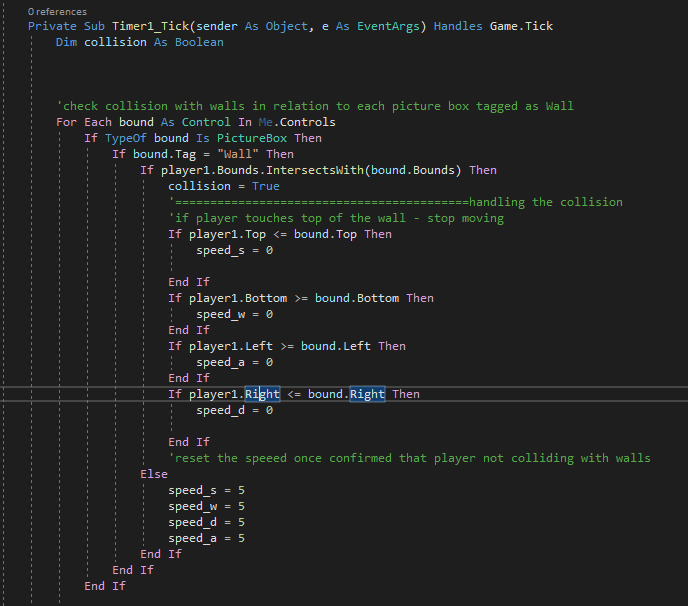
In order to make a game with walls and have my player travel around a map full of them, I must ensure that the player is unable to go through the walls as that would be unrealistic. In order to do this I implementedcollision detection into the update section of my code. Instead of applying conditions to each picture box I made, I tagged them all under the same name “wall” as they would be my walls, this would mean that I’d be able to refer to all of the picture boxes at once when writing my code.

The following code checks which side of the player is touching which side of the wall, this means that I can respectively restrict movement for the player in the direction that is touching the wall. For example if the left side of the player is touching the wall I would set the ‘d’ key movement speed to ‘0’.

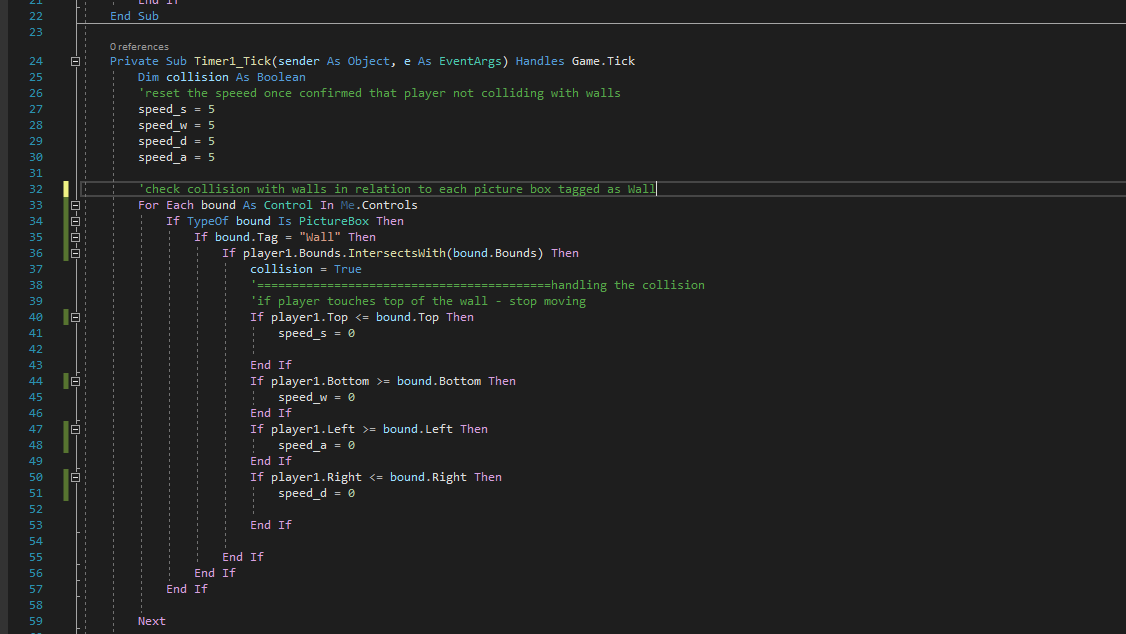


The code worked as intented and I now needed a way to return the movement speed back to normal. To do this I added an else statement after the if statement checking intersection with walls. The else statement would return all movement speed values to normal.

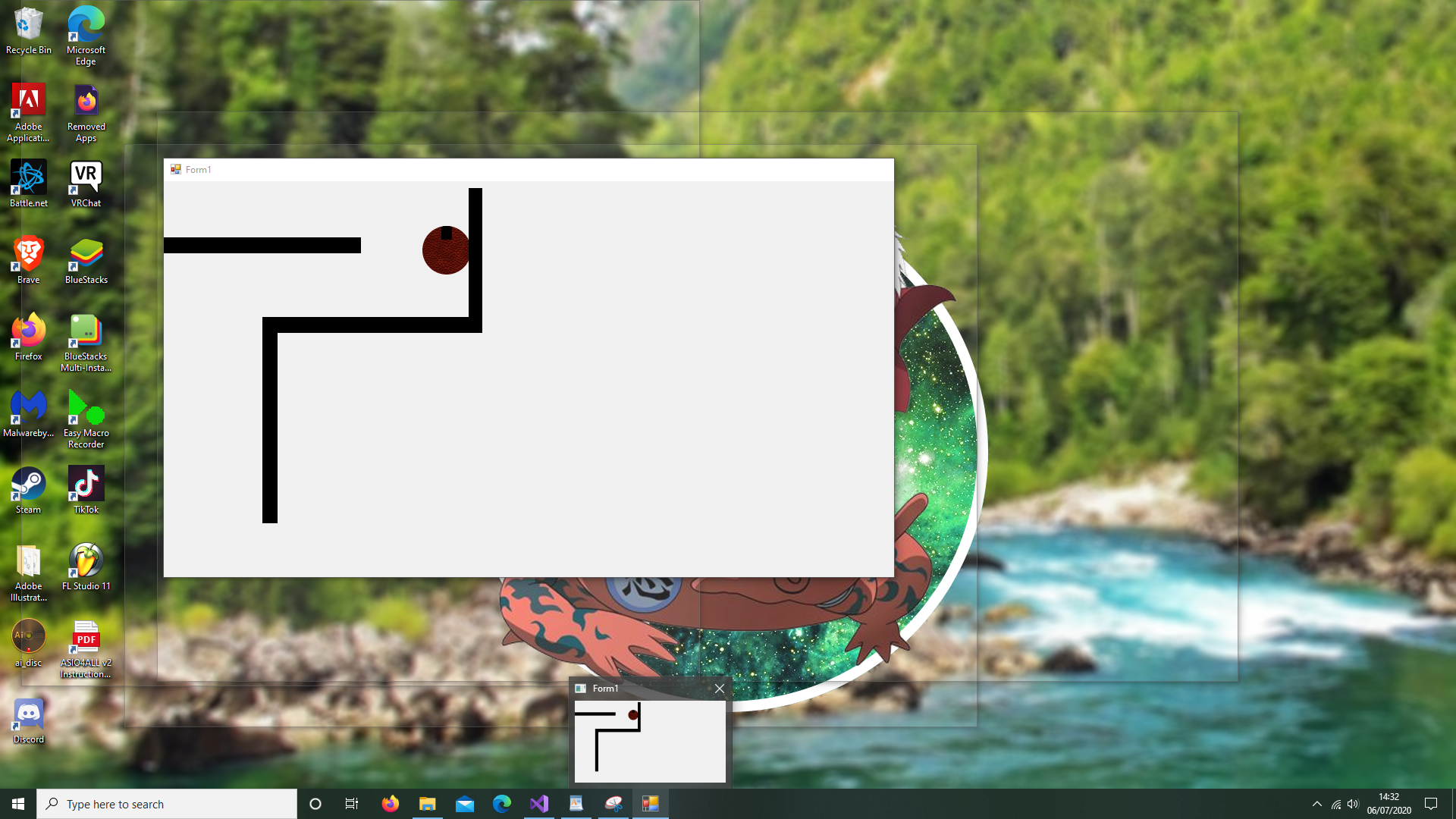
However, this didn’t work because as soon as the player’s movement was restricted the else statement would return it to normal immediatley.Therefore the player could move freely throgh the walls. The following picture shows the faulty code.



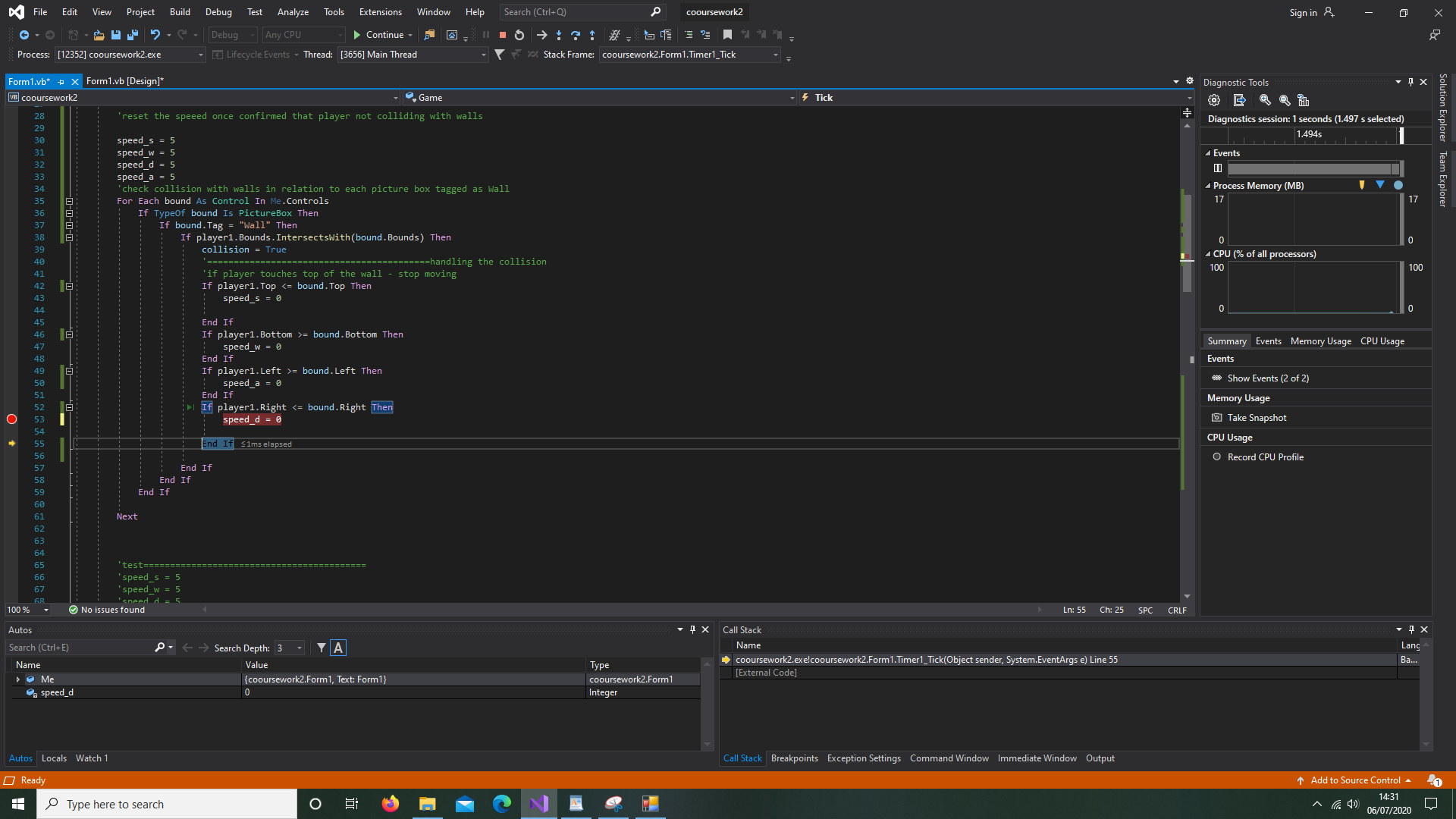
To fix this, instead of resetting the speed values inside the if/else statement, I did it before the collision check started. This ensured that when the player was not touching the wall it would be able to freely move and when it was touching the wall, its speed in the direction of the wall would be 0.



**REMEDIAL ACTION**



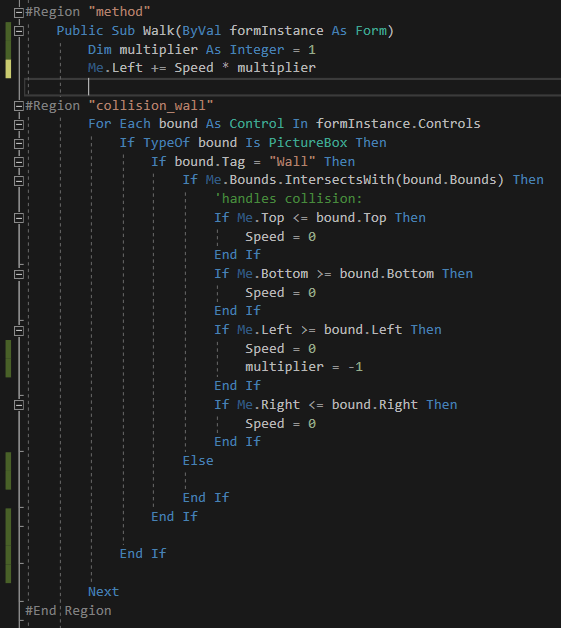
TEST



The speed now drops to 0 when touching the specified side.

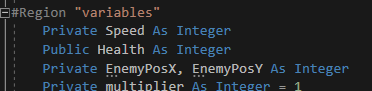
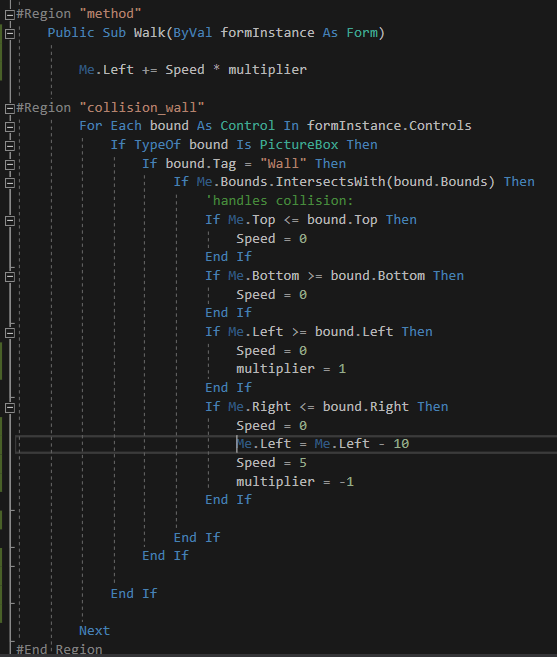
**Enemy won’t bounce off of walls**

Horizontal movement of enemy should be inverted by making the multiplier negative when it collides, however this wasn’t happening as the method was constantly being called from the timer hence it was re-declaring the multiplier and making it equal to 1.



Before

To fix this, I created the variables in the variables section of the class rather than the individual method. I also made it so that the enemy gets pushed away from the wall as soon as it touches it, so that it isn’t in a collision response loop. I also made it so that the speed gets set back to 0 after it’s pushed away from the wall so that it can still move.



**REMEDIAL ACTION**

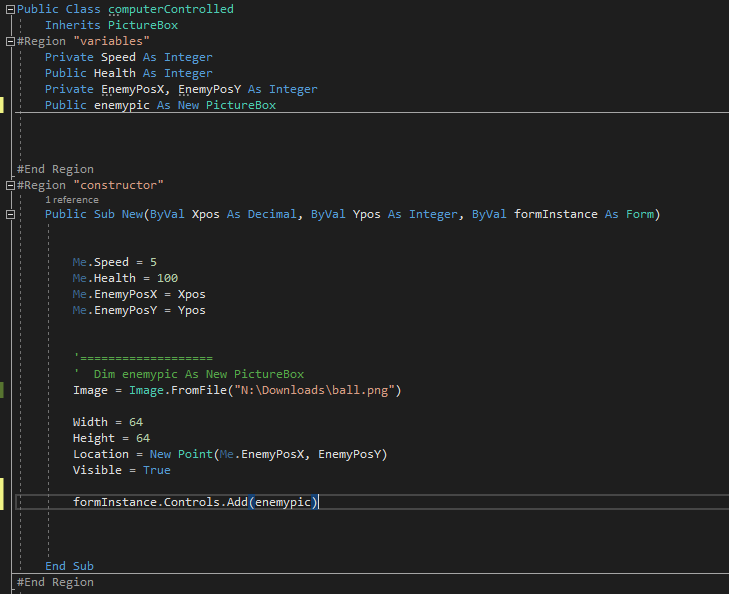
**Version 3**

**Picture boxes**

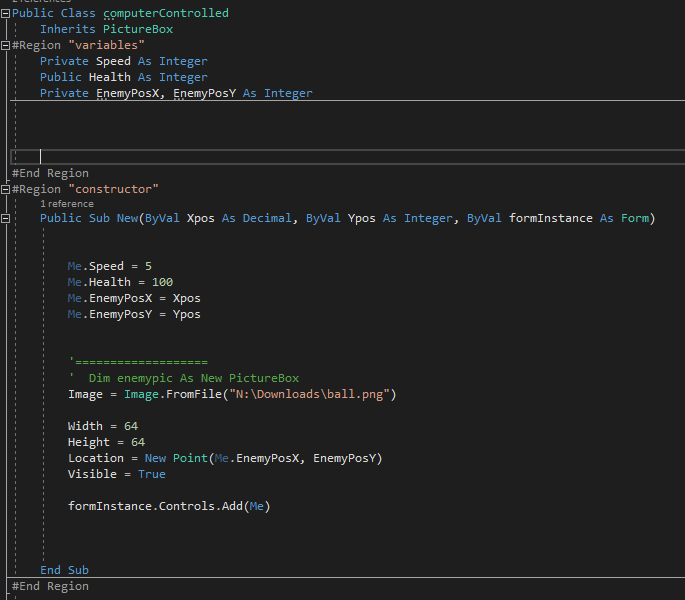
The computer controlled class that instantiates enemies will inherit from the picturebox class.

**Problem 1:**

The instantiated picture box stopped showing up when I made the whole class a picture box class by inheriting from the picture box class. The reason for this was that I was unnecessarily creating a picture box in a class that was already a picture box. To remedy this I removed the variable enemypic:



Before



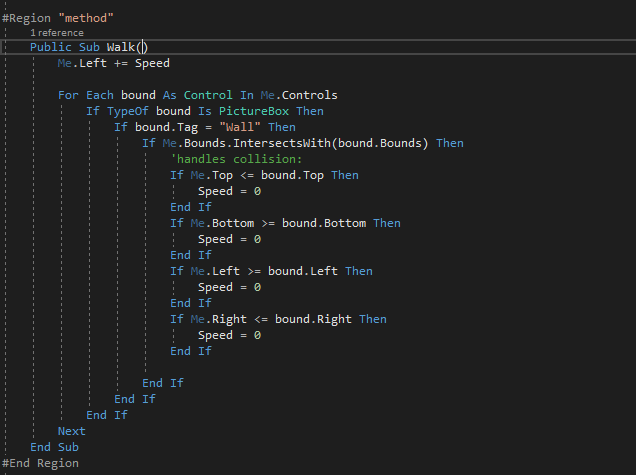
**REMEDIAL ACTION**

(changed formInstance.Controls.Add(enemypic)) to formInstance.Controls.Add(Me) )

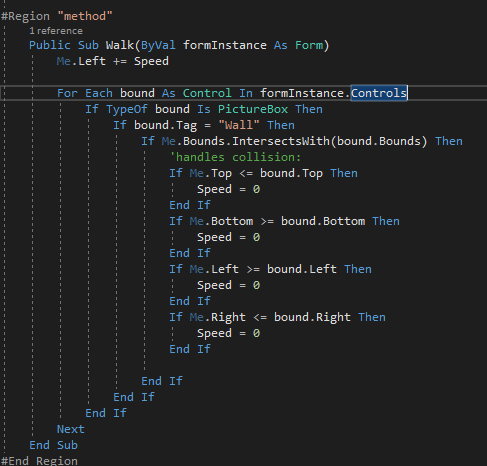
And removed top

**Problem 2:**

The dynamically created picture box was going through the walls rather than stopping upon collision. This was because when I was checking me.controls (which was meant to be for the main form) it was referring to the class when I said me. To fix this, I passed in the main forms’ form instance through the parameter and handled it accordingly.



Before



**REMEDIAL ACTION**

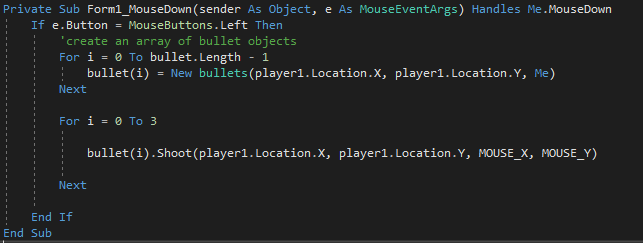


**Bullets**

The bullet class would inherit from picture box and would have a method called shoot that would be activated upon the mouse click. The idea was to create a bullet that would calculate the gradient between 2 points, being the player and the position of the mouse when it was clicked.

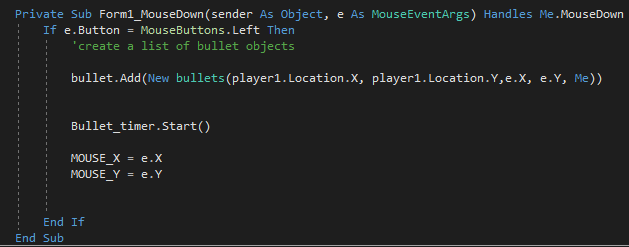
**Problem 1:**

The bullet would not move when the shoot subroutine was called.

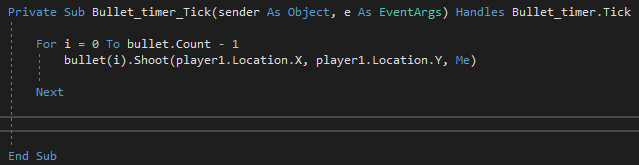


Before

I realised that upon clicking the mouse, when the shoot method was called, it would only execute it once so the bullet would move along by one unit. This was because the shoot method would only be called once.

I realised that I would need to implement a bullet timer to fix. The bullet timer would need to be started upon mouse click and it would call the shoot subroutine, this would mean that the bullet would definitely travel.

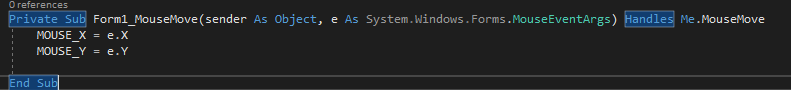
**REMEDIAL ACTION**



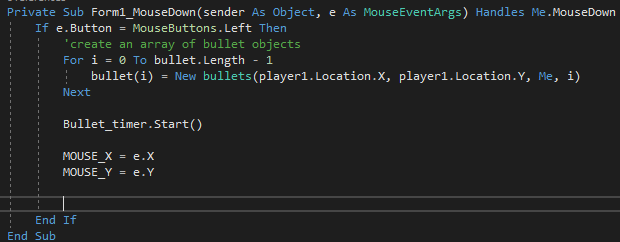
**REMEDIAL ACTION**

**Problem 2:**

When I clicked for the bullet to be shot, the bullet would follow the mouse rather than the position where it was last clicked. This was because I had been getting the mouse co-ordinates while the mouse was being moved rather than when it was clicked:



To fix this, I moved the code out of this event, into the MouseDown event so that the bullet would go to the last position where the mouse was clicked.



**REMEDIAL ACTION**

**Problem 3:**

The bullet would stop moving when a new bullet was created.

To fix this I changed it from an array to a list

Before

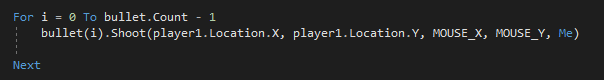


**REMEDIAL ACTION**



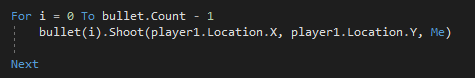
**Problem 4:**

The redundant bullets would follow every new mouse click despite not being able to move. This was because the new mouse position was being updated from the shoot method in the timer.

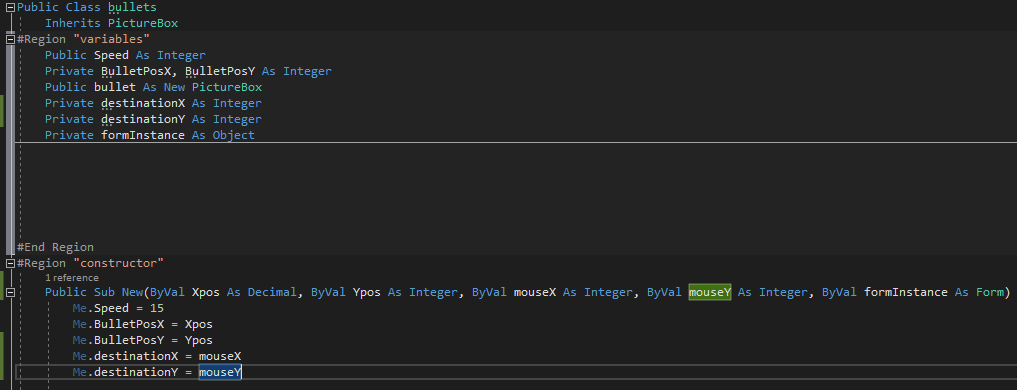


Before

To fix this, I removed its capability to update the mouse position. Instead, I encapsulated the destination of the bullet in itself.

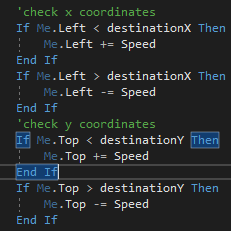


**REMEDIAL ACTION**



**Problem 5:**

At first, the bullet would reach the position that was last clicked by the mouse and stop. A real bullet would continue to travel until it collided with something. My previous code was making it reach the destination and stopping there:



Before

I wrote new code that would take a scale factor and apply it to the gradient that the bullet needed to move through. I did this by finding the distance between the player and mouse click and then divided the speed of the bullet (which I had previously set) by the distance between the two points. This would give me a scale factor to apply to the change in y and change in x to make them move at the speed I wanted.

**REMEDIAL ACTION**

Made in constructor:



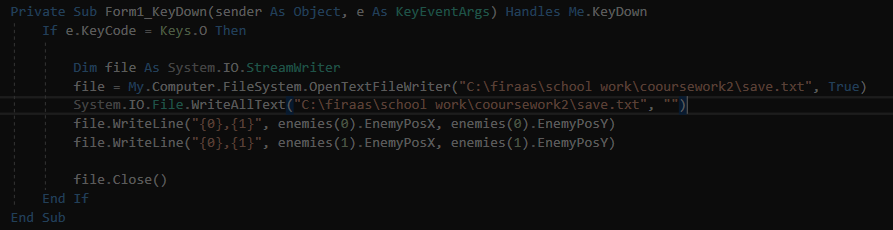
In shoot method:

**REMEDIAL ACTION**



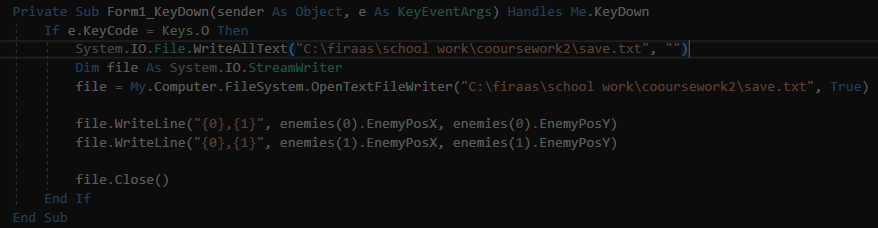
**Writing to files**

I was attempting to clear file before writing to it however I was notified with an error that said the file was already in use.



Before

It turns out that I had to use the line of code that I was using to clear it before I opened the file in the program.



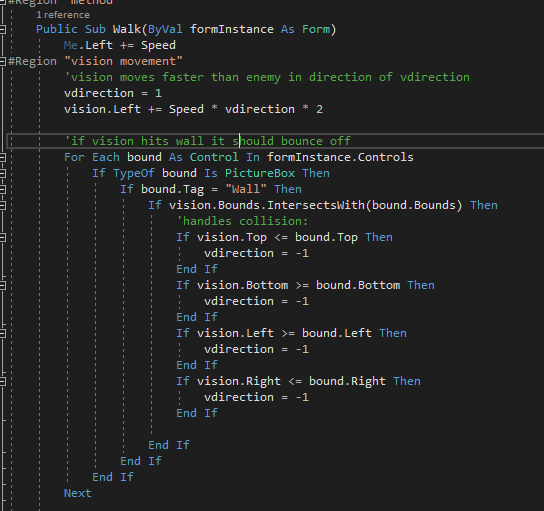
**REMEDIAL ACTION**

**Vision**

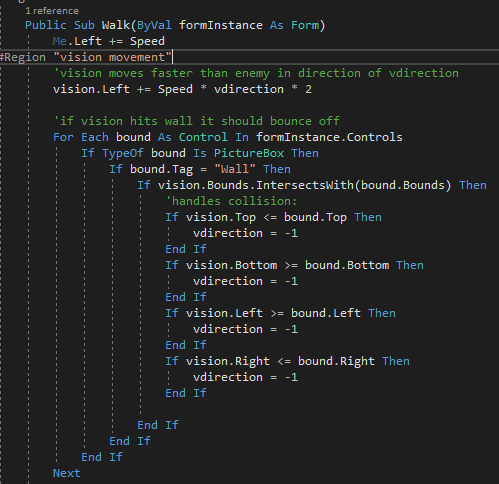
To give the enemies vision, I would have an invisible picture box shoot towards the player and if it collided after a certain distance, then the player was in the enemy’s field of view. I went with this approach because it was more intuitive to implement in my game as it was not created on a grid. Had this been based on a grid, I would have used a ray casting algorithm and Bresenham’s line algorithm to work out exactly where the player was in relation to the enemy. This algorithm would have been more memory efficient.

**Problem 1:**

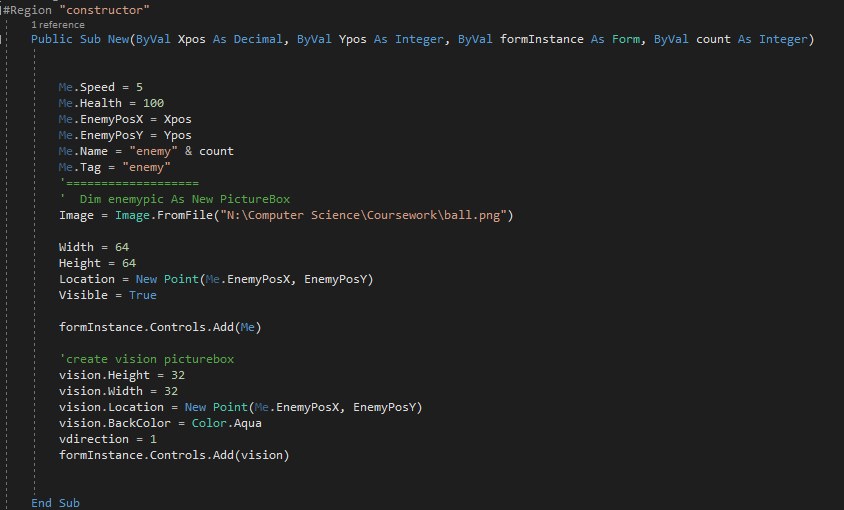
To test if the vision would bounce off walls I set the direction to change upon colliding with the walls however, the vision would continue moving in one direction. The issue was that the timer would continuously call this method and the ‘vdirection’ would be set as 1 upon each call so as soon as it was changed to -1 it would get reset to 1 right after and didn’t cause any change as a result. To remedy this, I set ‘vdirection’ to 1 in the constructor rather than the method of the class.



Before



**REMEDIAL ACTION**

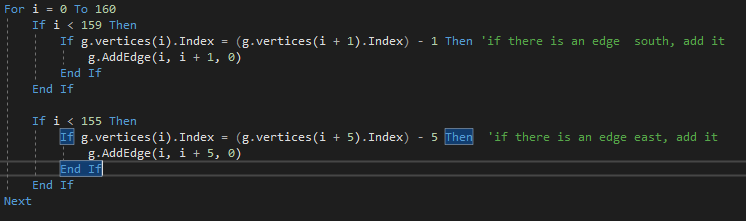


**REMEDIAL ACTION**

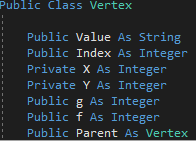
**Astar Path finding**

The reason that I chose to use the A\* path finding algorithm over its predecessor, Dijkstra's algorithm was due to its implementation of heuristics. While Dijkstra's algorithm finds the shortest distance between 2 points, it implements a breadth first traversal. Therefore, it explores every node and ends up using a lot more memory and processing than necessary. The A\* uses a g score, f score and h value. The g value is the cost given to each node, the h value is the heuristic that’s calculated and the f score is derived by adding both g and h values. I use the Manhattan heuristic to help determine which path to follow next. Furthermore, I used an adjacency matrix rather than an adjacency list to create the graph. This decision was made as I thought that I had more information available online on adjacency matrixes than adjacency lists.

The vertices would be made with a nested loop creating a node in intervals of 50 across a grid. The edges would be added to the south and east of each node if there was a node present at that location.

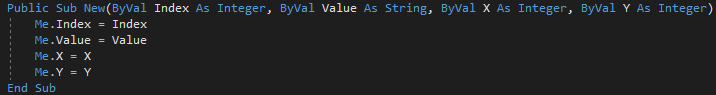


There would be a separate vertex class from which vertices can be instantiated. The following attributes would be assigned:



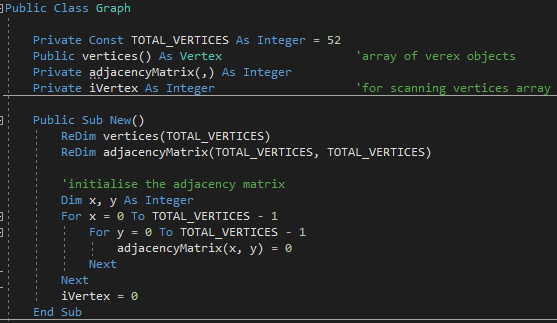
* Index is how each vertex will be uniquely identified
* X and Y are the grid coordinates for each vertex
* g is the path distance from the start
* f is the value calculated by adding the g value and heuristic
* Parent identifies the previous vertex

The constructor has values for the index, value and co-ordinate positions passed in:

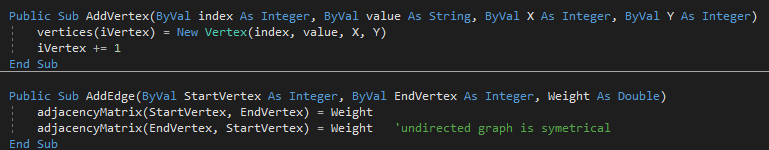


The heuristic function returns a h value for each node. It is calculated using the Manhattan distance between the current vertex and the destination vertex.

The graph was also created in an object oriented fashion:

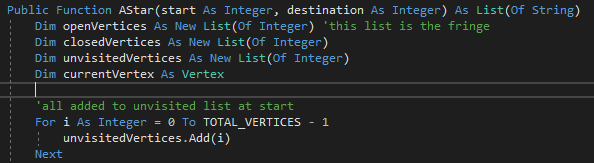


To add vertexes and edges to the graph, a method was created in the graph class:

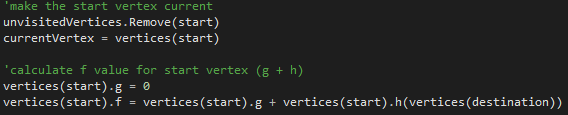


As the graph is undirected, the adjacency matrix would have to be symmetrical.

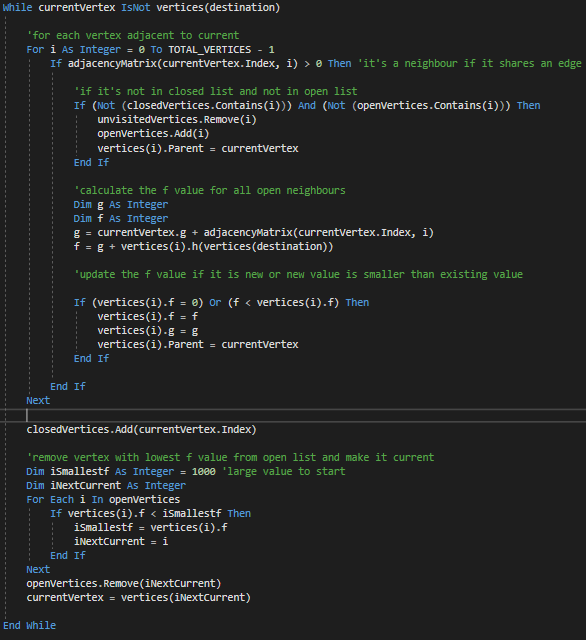
To start off my A\* function, the unvisited vertices list was populated as nothing has been visited yet:



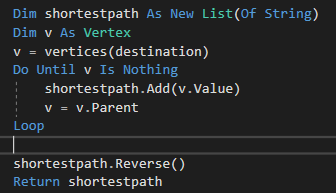
The start vertex would be made current by adding it to the current vertices and removing it from the unvisited list and the f value would be calculated by calling the h function along with getting the g value for the start node:



My main A\* algorithm works with a while loop until the destination is found:



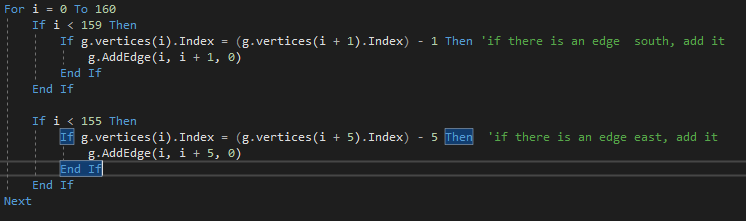
The shortest path information is stored as a list and returned:



The list is reversed as it is added to in the order from the destination rather than the start.

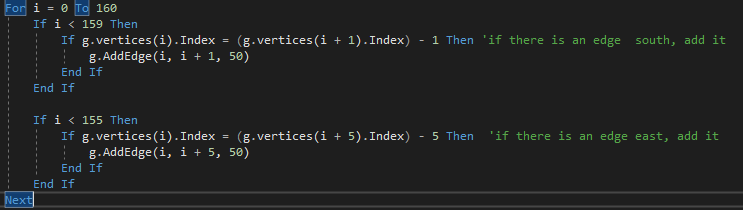
**Problem 1:**

The Astar function would crash the program as it would be stuck in an infinite loop of adding nodes to the closed vertices list. I found that this was the issue because the weights had been incorrectly set to 0 and hence, there was nothing for the A\* algorithm to work with to return the shortest distance and would continue to loop.



Before

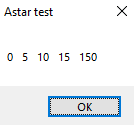
Weight being set



**REMEDIAL ACTION**

Remedial action:^

Test:



TEST

# Testing

**Post development testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No** | **Description of test** | **Test Method** | **Expected Result** |
| 1 | Movement of player | Use WASD to move the player | The player should move Up, left, down and right respectively. |
| 2 | Enemies Should move in their default move cycles or remain static if they’re assigned to do so. | Run the program | The enemies should move accordingly to how they were initially assigned to move. |
| 3 | Collision | Player – walls: Have each side of the player sprite move into a wall.  Player – enemies: Move the player to touch the enemies  Bullet – walls: Shoot a bullet into the wall  Bullet – enemies: Shoot bullet towards enemies | Player – wall: player’s movement should be halted when touching the wall.  Player – enemies: player should lose health.  Bullet – walls: bullet should disappear and be disposed of.  Bullet – enemies: enemies should die (both bullet and enemy should be disposed of). |
| 4 | Player Death/Damage | Let the player collide with enemy and wait for the health reach 0. | Player should die and a message should be displayed accordingly. |
| 5 | Shooting bullets | Click at any point on the screen to shoot a bullet – test every cardinal direction around the player to ensure that there’s no issues with the position vectors. | A bullet should gradually move towards the last position clicked by the cursor at a constant speed in a constant direction. |
| 6 | Saving | Click save to save the game. | The coordinates of all sprites should be copied to a text file. |
| 7 | Loading | Click load to load a game. | The coordinates from the text files should be read and loaded into the game. |
| 8 | Vision | Have the player move past all of the cardinal directions of the enemy to ensure that the enemy can detect the player in all valid circumstances. | The enemy should begin the path finding algorithm and break from its standard move cycle to follow the player. |
| 9 | Path finding | After the enemy detects the player, see if it follows the player up, down left, right and around walls. | The enemy should follow the player wherever they may decide to go. |
| 10 | Record scores | Click on the scores button. | Score should be displayed. |
| 11 | Crash prevention | While another application is using the save file, attempt to save the game. – A ‘System.Io.IOException’ error should be received which means that the file is already in use. | The game shouldn’t crash; rather it should prompt the user that the game cannot be saved at the moment. A crash would result in the player being dis satisfied as they would lose their progress which is the exact opposite of what they were trying to achieve. |
| 12 | Pausing | Press the pause key – ‘p’ twice. | The game should freeze, no objects should move and the score shouldn’t be incremented. When the key is pressed again, the game should resume as normal. |

**Evidence of post development testing**

* Video 1 shows that movement in all directions is functional.
* Video 2 shows that enemies are following their default move cycle.
* Video 3 shows that collision worked between: player-wall, player-enemies, bullet-wall, bullet-enemies in all four of the cardinal directions.
* Video 4 shows the player taking damage and dying.
* Video 5 shows that bullets were successfully fired in all directions.
* Video 6 shows that the coordinates of the enemies and players’ location had been successfully copied to a text file.
* Video 8-9 showed that the Vision and Path finding algorithms were unsuccessful.
* Video 10 showed scores that were recorded being displayed from a text file.
* Video 11 showed the program trying to save and write to a file while that file was already in use. It successfully displayed an error message rather than crashing the program.
* Video 12 showed the game being paused and un-paused numerous times.

**Stakeholder questions:**

I had decided to ask quantifiable questions as they are easily measured. This means that if I ever improve upon the solution, I can carry out the same questionnaire and my results can be compared without any ambiguity and they may even get plotted on a graph.

|  |  |  |
| --- | --- | --- |
| **Question** | **Justification** | **Answer** |
| How would you rate the user interface from one to ten? | This will help me understand if the user had any trouble with navigating through the game. | 9 |
| On a scale of one to ten how likely are you to recommend this game to a child? | Based on how the user will score this, I can gauge their honest opinion on how suitable they think the game is for children. | 9 |

# Evaluation

**Success criteria**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Requirements** | **How to test this** | **Justification** | **✔** |
| 1 | The game should have a pause function | Check by pressing the button ‘p’ which will cause a pause screen to appear in front of the player, giving them options to save the game/quit. | A pause function will allow the user to take breaks by giving them an option to save progress and come back to it later, it will also allow the player to tweak settings while playing the game. | ✔ |
| 2 | Crash prevention – the game shouldn’t crash while the player is playing the game | Enter erroneous inputs where possible, they shouldn’t lead the game to crash. | Game crashes can lead to players losing their progress which is not ideal as they would have wasted their time. | ✔ |
| 3 | Have a save file - data about the player’s current status can be exported to a text file that can be later read when loading a save. Things such as location score and inventory will be recorded. | Check the text file after the save has been made to ensure all of the data has been exported properly. | A save function is needed so that the player can later pick up from where they left off by saving their progress to a specified location. | ✔ |
| 4 | Save files should be loaded | A text file will be read and the game will load the information on the screen accordingly e.g. last position and what was previously in the inventory. | It will allow the save files to be loaded and the player can resume from where they left off. | ✔ |
| 5 | The weapons will be fired in relation to the position of the cursor - the position of the mouse must be constantly recorded and a crosshair will follow. When the player decides to fire, the attack will begin its course to the last place the cursor was when it decided to fire. | Check by shooting a bullet and seeing if it reaches the cursor. | This will allow the player a visual representation of where their attacks are going and as a result, they can aim their attacks in order to kill enemies. The mouse is an effective way for the player to point out where they want to attack as swiftly as possible. | ✔ |
| 6 | ‘w’, ’a’, ’s’, ’d’ will control the players movements. | ‘W’, ’A’, ’S’, ’D’ will cause the players sprite to move ‘up’, ’down’, ’left’, ’right’ respectively. | The player will be able to confortable use the mouse while moving the player as the directional controls will be on the far left side of the keyboard whereas the mouse is typically on the right side. It is ergonomic and generally, the player should be most comfortable while playing in this position. | ✔ |
| 7 | Enemies will have ‘vision lines’ - unlike 3d vision cones, this is a 2d game so lines will be used. If a player enters the enemies’ vision line they should be alerted of the players position and the path finding subroutine should be executed. | The player should move past the enemies’ un hindered line of sight and the enemy should detect it. | This gives the game an element of stealth as it objectively pushes the player to avoid the enemy line of sight. It has to detect the player in order to pursue them. | ❌ |
| 8 | Enemies should track the player - when the enemy is alerted of the players whereabouts (from their vision), they should apply a path finding algorithm to approach the player. They should be able to find the best way to reach the player given their current positions. | After the ‘vision’ (the player has been spotted) has been confirmed, the path finding function should calculate the shortest distance and pursue the player. | This will give a more realistic feel to the game if the enemy can track the player once alerted. | ❌ |
| 9 | Clear user interface: display items/stats - the game should show remaining ammo and health at the top of the screen, as it is used up the number should decrease accordingly. The weapons in use should also be displayed in a clear fashion. | This can be measured post testing when the stakeholders have had a go at the game. They can fill out a questionnaire on how clear the UI was. The questionnaire will be of quantitative values as it is the most efficient way to measure results. | This gives the player information and lets them know if they should save their bullets/health and whether they should play conservatively. Without a clear UI, players may have a hard time navigating the game or just playing it in general. | ✔ |
| 10 | Collision - the player should collide with bullets/walls and an action should be carried out accordingly. | All four cardinal points of the picture boxes will be tested against all of the cardinal points of the other picture boxes that may have a chance of colliding with the player. | Perfect collision ensures that the player doesn’t fall out of bounds. It allows the game to check if the player is interacting with the environment e.g. being shot and will act accordingly e.g. take damage. | ✔ |
| 11 | Must be child friendly | Have the stakeholder fill out a questionnaire on whether the game brought about any violent tendencies or caused the user discomfort. They can rate how comfortable they felt playing the game on a scale of 1 to 10. | As this game prioritises the wellbeing of children, it is essential that the stakeholder deems it worthy of being played by children. | ✔ |
| 12 | Variety of weapons should be fun to use | Have the stakeholder try all of the weapons and see if they found themselves switching between the weapons a lot. | Frequently switching between weapons indicates that they are each unique in their abilities and play style. The added layer of just switching between guns mid game would make the game more interesting. | ❌ |

The bulk of my shortcomings can be found in the ‘Pac-man’ style chase sequence that was intended to be implemented in the game. While the enemy sprites do move on their own, according to their predetermined patrol routes, they won’t follow the player. Success criteria 7 and 8 were not able to be met. This is because they both relied upon a grid like structure to be able to function. While I was able to program the A\* algorithm, I wasn’t able to integrate it with the game as I wasn’t able to figure out a way to dynamically create a grid and have the A\* algorithm work with it. This inability partially comes from the time constraint that was felt during the final stages of development. If more time was available, I may have been able to carry out further research on how to program such an algorithm and would have been able to easily meet both criteria’s as a result.