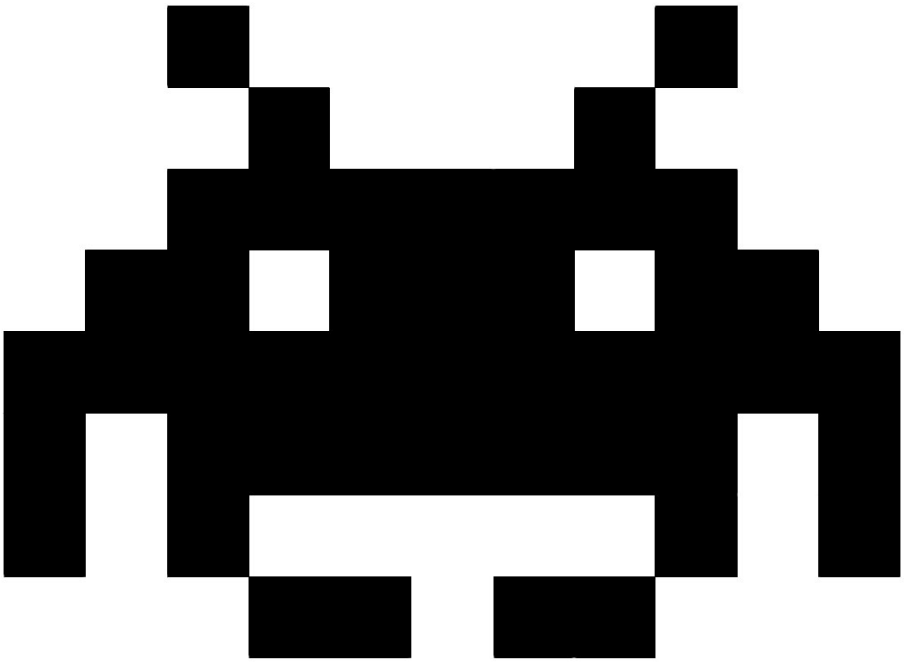
Interim Report

Creating a Multi-Platform Space Invaders Clone with Advanced Features Using Modern Hardware and Programming Environment



Written by Stephen McQueen, Supervised by Dr F.C. Langbein and Moderated by Prof R.R. Martin

Creating a Multi-Platform Space Invaders Clone with Advanced Features Using Modern Hardware and Programming Environment

Abstract

This report intends to introduce and discuss my project of implementing a classic arcade game, namely Space Invaders. I will introduce you to the game that was at the forefront of the video games technology and video games industry of which today is worth $10.3 billion in the USA alone. I will also discuss in detail the challenges of such a project and what modern game technologies now exist that can aid me in tackling them as well as improving the original game.

1. Introduction

This aim of this report is to demonstrate the results produced at the half way stage of the overall project lifecycle. As discussed in the initial plan of the project, the intended deliverables are as follows:

- Outcomes of research
    - Space Invaders
    - Visual Studio and the XNA Game Studio framework
    - Game AI
    - Hardware Platforms
- Project Analysis and Approach
    - Breakdown the project and discuss feasible milestones for project progression
- Game Design
    - Undertake the design stage of the project

Therefore I have structured this report in such a way as to reflect the above deliverables achieved thus far. This has been done by dividing the report into logical sections namely:

1. Introduction
2. Background
3. Development Tools
4. Existing Games
5. Game Technologies
6. Specification
7. Design
8. Conclusion
9. Glossary

For each of the above sections I have then used sub-sections to discuss specific topics in particular detail.

2. Background

Space Invaders is an arcade video game released in 1978, Taito, Japan and was one of the first shooting games ever made. The game itself is simplistic in nature which is due to design and hardware limitations at the time. In fact the microcomputers that were around were not powerful enough to process large data sets and perform complex operations on them. As a result the lead designer (Tomohiro Nishikado) had to design custom hardware and development tools before implementing anything.

Nishikado settled on using the latest American processor Intel 8080 with a clock speed of 2MHz. Even with this hardware it was not powerful enough to render the graphics to the screen in colour without impacting the games performance. However despite all of the hardware limitations present, Space Invaders quickly became one of the most influential video games of all time. To date the game currently holds 5 Guinness World Records including, “Longest Running Video Game Series” and “Longest Running Role Playing Game Franchise”.

The main objective of the game is for the player spaceship to defend their home planet and destroy never ending waves of increasingly difficult aliens with their laser canon. The enemies move together as a group from one side of the screen to the other (horizontally) in a stuttering fashion. Once the enemies on the side closest to the edge of the screen actually touch the screen boundary they all move forward (vertically) together slightly before moving towards the other side of the screen (horizontally). The player gets 3 lives to achieve the highest possible score by defeating as many enemies as possible without being killed themselves.

2.1 My Project

As stated in the project title my aim is to recreate the classic Space Invaders game using modern programming languages and hardware and discuss the challenges I faced in comparison to those found by Tomohiro Nishikado. This will be interesting project due to the vast advancements in technology since the release of Space Invaders. For example the processing power of the CPU in my mid range smart mobile phone is 1GHz which is astonishingly 1,000 times faster than the Intel 8080 CPU found in the original Space Invaders game.

More importantly however my secondary aim is to add more elements to the original game, such as more intelligent enemies, networking and power ups and discuss the feasibility of these.

3. Development Tools

3.1 Hardware Overview

As shown in the report title, it is important for the game produced at the end of the project is compatible with as many hardware platforms as is feasibly possible within the time frame provided.

Since the release of the original Space Invaders there has been a massive increase of available hardware platforms for games including bespoke hardware known as “consoles”. Consoles include; Playstation’s 1, 2 and 3, Xbox and Xbox 360, Nintendo Wii, Gamecube and many more. More recently powerful handheld consoles have become available such as the Nintendo Gameboy DS and Playstation Vita. However in today’s environment more and more non gaming specific hardware is more than capable of processing through games such as most mobile phones (especially the more powerful smart phones) and of course the traditional personal computer.

Therefore it is clear that there is a plethora of available platforms in which to create this game for. However as the focus is on modern platforms this narrows down the list slightly. In order to achieve as much gaming diversity as possible, ideally I would like the game to run on a personal computer, a games console and a hand held platform.

3.2 Modern Programming Languages

Since the late 1970’s there has been a large increase in available high level programming languages such as Java, C#, Objective C, Python, Flash, Lua and Visual Basic. Also several Integrated Development Environment’s (IDE’s) have become available over the years which make the creation of complex programs more organised and intuitive for software engineers.

Being most proficient in object oriented languages, originally Java appealed to me the most, mainly because I have spent the past few years of my degree learning it. Not only that but it is the language used by the Android mobile operating system for which there is a free Software Development Kit. This appealed to me greatly as it would allow me to create a multiplatform game for two very different hardware platforms namely a personal computer and Android mobile phone.

However I also came across a free IDE called Visual Studio 2010 Express Edition for Windows Phone (with Windows Phone Emulator) by Microsoft which allowed applications to be written for Windows Operating System (OS) PC's, the Xbox 360 console and Windows OS mobile phones. Visual Studio also offered a variety of programming languages such as Visual Basic, C++ and C#. After doing some brief research it became apparent that there is a specific framework available for game programming called XNA Game Studio 4.0 (current version) which provides a vast set of tools that could aid me in this project.

Therefore despite not having any programming experience using C# it seemed to be the most appropriate choice due to the vast resources available with the Visual Studio IDE and the ability to create applications for three platforms in one place.

3.3 Supported Platforms

3.3.1 Xbox 360 & Windows OS PC Platform

As discussed previously the Visual Studio IDE allows for programs to run on the Xbox 360 console and Windows OS PC's. It is worth noting that the above hardware is almost limitless in its ability with regards to the expansion of an original 1970's game. For example a modern computer will house 2GB of DDR2 RAM, expansive hard disk space averaging at 500GB, dual/quad core CPU's averaging at 2.5GHz as well as Graphics Processing Units capable of speeds of 1GHz for dedicated graphics computation.

It is also a similar story with regards to the Xbox 360 games console which has been purpose built for running complex games with ground breaking graphics whilst maintaining consistently high frame rates in high definition. Therefore it is worth noting that my rendition of Space Invaders will be in a high definition resolution 1280 x 720.

3.3.2 Windows 7 OS Mobile Phone

The main difference here is that the Windows Phone is far more limited from a hardware perspective as it is a hand held device. However despite the lack of hardware power with regards to the above, the Windows Phone does have a far greater screen resolution (as is often the case with touch screens) and therefore this version of the game will be in a higher resolution than that found in the Xbox 360 and PC version of the game. Therefore this will have to be considered when designing the game to check if any advanced feature will be too computationally intensive for the hardware and whether a less intensive solution can be found.

Another obvious difference is the main purpose of the hardware is a mobile phone and therefore this brings several other considerations to be taken into account when designing the game. For example the game can be interrupted by several things such as; an incoming phone call, text message, email, notification from social networking sites etc.

4. Existing Game Versions

4.1 Original Space Invaders

In order to gain some understanding of what aspects of the original Space Invaders I would have to implement, I firstly had to play through the game several times. From doing so I would hope to gain a clear perspective of the games objectives.

Before playing the game the player is presented with a list of enemies along with their scoring value so that you are aware of the scoring system. The player then has to laterally manoeuvre across the bottom of the screen to fire lasers at the pack of enemies to destroy them. The enemies follow a pattern of movement which involves moving from one side of the screen to the other (horizontally) but when a side is reached they advance towards the player (vertically) slightly before moving to the other side of the screen (horizontally).

From studying the game in detail I was able to determine its pros and cons in an effort to reveal any areas that may be improved. The results from the study are as follows:

Pros:

- Simplistic game design
- Gives the player a clear objective
- Effective user interface
- Competitive/Addictive

Cons:

- Repetitive game play
- Enemies do the same thing over and over again, the only thing that changes is the rate of movement and firing at the player.
- No variety present in game progression. No new elements of the game are introduced which can cause the player to react differently.
- Graphics are poor and very little use of animations
- Lack of Artificial Intelligence

After conducting this initial research I then looked at different versions of the game and evaluated their strengths and weaknesses. The aim of this was to see what others have done with the original game to adapt it to run on modern platforms and to find some inspirational ideas as to what additions I could make to the game to improve it.

4.2 Space Invaders 5

This game is a free version of the original game released in 2003 and is playable from www.funnygames.co.uk. The reason I did not look at evaluating Space Invaders 2, otherwise known as Space Invaders Deluxe is because the only real change made from the original was adding real colours by using RGB screens.

Space Invaders 5 adopts all of the concepts found in the original Space Invaders namely, the player has to destroy as many enemies as they can with a finite number of lives. However, this more modernistic interpretation of the game makes use of high definition graphics (but still in 2 Dimensions), background music and animations but it stands firm with the original game play concepts. It does make one slight alteration, in that the player fires missiles instead of lasers, but

nonetheless it wholly encapsulates the original game but with a new look as opposed to the actual game play.

Therefore after studying this version of the game (which was not released for sale), I came up with the following pros and cons:

Pros:

- Improved graphics and animations
- New and more colourful user interface

Cons:

- No real evidence of enemy AI
    - It should however be noted that it does appear that the enemies do seem to be firing based on some form of calculation based on the players direction. For example the player is killed relatively quickly if left stationary which seems to suggest that this calculation is occurring.
- No new game elements introduced

Therefore it seems that despite the advancements in technology since the original game it appears that the emphasis has been on improving the game graphically. This does however make sense as Graphic Processing Units (GPU) today are almost as powerful as olden day CPU's and can allow for highly detailed graphics without affecting the games overall performance. However it is also fair to assume that the designer of this version of the game may have intended to stay as close to the original game as possible.

It is also worth noting that there is a large amount of similar games based on Space Invaders present on this website (www.funnygames.co.uk) which clearly reflects its popularity among programmers to show off their ability and creativity. However I have not analysed any of these as most have strayed too far from the original scope of the game.

4.3 Galaxian

Galaxian is a very similar style of game that was released in 1979 by Namco a year after the release of Space Invaders. The game had a very similar objective, namely defeat never ending waves of aliens with your space ship and its laser canon. However the main difference is that of how the enemies go about destroying the player. As opposed to marching from one side of the screen to other and progressing down the screen, the Galaxian enemies march from left to right but frequently break away from the main group to fly in an arc like motion towards the player. This therefore adds an interesting twist to the game play, as now the player not only has to focus on avoiding incoming fire from the main group of enemies but they must also try and evade/destroy enemies that are hurtling towards them.

Of course it would seem this would require some form of Artificial Intelligence however, again the game was limited by the hardware of its time and so the methodology behind this would be as simplistic as possible as to avoid performance issues. My findings from evaluating Galaxian compared to Space Invaders are as follows:

Pros:

- Improved graphics
- Different element of game play
- Enemies are no longer linear in movement and can break away and attack the user head on.

- Power ups allows the player to adopt a different style of game play
- The player can destroy enemy lasers with their own (upon collision)

Cons:

- No use of barriers
- Enemies do not progress forwards towards the player (so no sense of impending doom) and just move from one side of the screen to the other

5. Possible Advancements/Modifications

Before undertaking detailed design of my Space Invaders game I would first need to consider what aspects of the game I wish to improve on and implement as advanced features. As a result of the above research I came up with the following possibilities:

- Artificial Intelligence
- Collision Detection
- Networking
- User Interface

5.1 Game Engines

Before discussing any of the above in any detail I firstly had to decide on a relevant game engine architecture that I can use as the skeletal framework of the project. This would be a crucial decision as the framework will affect the programming style and efficiency of the game. With this in mind I undertook a quick study into different game engine architectures and found the following information.

5.1.1 Monolithic Architecture (Unreal Engine)

This type of architecture is extremely popular with bespoke shooting games such as the Unreal Tournament game franchise which implements the Unreal Engine. As a result this type of engine architecture is often very genre specific and can take a lot of time and effort to create from scratch.

5.1.2 Modular Architecture (C4 Engine)

This is a more modern approach this is more often used by game engine developers due its object oriented approach. As a result it is easier to expand upon original functionality by creating new objects etc.

5.1.3 Tool Kit Architecture (jME)

This is an example of an open source game engine which also implements a highly object oriented approach and is designed for ease of expansion.

5.1.4 XNA Game Studio Architecture

However considering the above it is most appropriate that when creating a new “XNA Project” in Visual Studio 2010, XNA provides a backbone framework which you can add too. It is noted that this is not a game engine as such, but it does provide the user with three main game loops namely; initialise, update and draw. The update and draw methods are automatically called 60 times per second (thus achieving an average frame rate of 33.33 frames per second. This is more than likely

due to hardware limitations on the Windows Mobile Phones and so can be altered for Xbox 360 and PC versions of the game if needs be.

Therefore having considered the above architectures it makes logical sense to make use of what XNA has provided me with as the project aim is to recreate the original game and not to design a game engine to run it as this would be counterintuitive to my main objectives.

5.2 Game Artificial Intelligence

So far my project research has shown that the original arcade games lacked artificial intelligence. As discussed previously this was due to the severe hardware limitations and so the designers of these games were forced to make shortcuts in order to mimic "intelligent" behaviour in their Non Player Characters (NPC's).

Consequently this gave rise to a completely new branch of Artificial Intelligence and this was quickly branded as "Game AI". Game AI has the simple aim of "if the player believes the agent they are playing against is intelligent, then it is intelligent". Therefore using this aim I too can take certain shortcuts to create the illusion of intelligence if needs be. However Game AI has seen a surge of popularity in recent years due to its increasing complexity and highly demanding customers.

5.2.1 Machine Learning

I had the idea of the enemies reading and predicting the player's actions early on during the project conceptualisation, but I asked myself why, and what would this achieve? Well this is because I put myself into the enemies shoes (or spaceships in this case), and thought what would I do to kill the player? The answer I came up with was predicting and manipulating the player's movements. For example if the player was moving in a certain direction, let's say left, then I as an enemy would attempt to fire a laser beam further to the left in the hope it would intersect their flight path and destroy them. However as my opponent is intelligent I can assume that they will take evasive action and as such I would then fire once more to the right of the original laser beam in the hope it collides with the player.

Therefore this suggests that in order to make the enemies intelligent as such, they will be required to "predict" the player's movements or even manipulate them to their own dastardly needs. In order to accomplish such a feat it would be required for the enemies to learn over time and this is called "machine learning". Machine learning can be defined as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"[1].

5.2.2 Bayesian Networks & Probability Engines

In order to achieve machine learning I would need a method of collecting relevant data about the player, processing it and achieving an appropriate outcome. Bayes Theorem is a highly popular statistical technique used to determine the probability of an event occurring given a relevant piece of information (P (A|B)) and is frequently used in spam filters to determine the probability of an email being spam.

However ideally in a video game I will need to determine a probability given a data set of known relevant information and thereby the use of a Bayesian Network would be more applicable. The formula for a Bayesian Network is: $P(G,S,R) = P(G|S,R)P(S|R)P(R)$. This would be ideal as it could provide the probability of the player reacting in a certain way given a certain scenario.

Therefore for this idea to work efficiently I will need to create some form of a probability engine that can quickly produce a probability when given a data set. There is also some positive supporting evidence shown in the paper "Bayesian Networks in Computer Games" [2] which illustrates that given the correct thresholds a Bayesian Network can lead to more intelligent NPC's over time. This can also be supported by games such as Forza Motorsport whereby the NPC's difficulty will depend on the player's current performance.

5.2.3 Finite State Machines

A Finite State Machine (FSM) can be defined as "a device, which has a finite number of states it can be in at any time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state can only be in one state at any moment in time" [3]. However it is not so much the machine that is useful to programmers but, the method in which they solve problems using "states". States are an extremely powerful way to represent behaviour in NPC's and are easy to manage and debug when implementing.

To give a basic example of a FSM imagine a NPC athlete getting ready to take part in a race. The NPC will have three basic states such as warm up, race and cool down. During each state the NPC will do something different and when the situation changes it can transition to a different state. Therefore the NPC will begin the event by warming up which will involve some light exercises and stretching. When the NPC is sufficiently warmed up it will transition to the race state which will mean it will head over to the track ready to compete. This state will provide all of the instructions and knowledge required for its specific event in this case it's the 100m sprint. Therefore the NPC will be aware that it will be allocated a specific lane in which to run, to run after the starting gun and where the finish line is. When the NPC has finished the race it will transition to the cool down state in which it will pant frantically trying to recover its breath and eventually do some more stretches.

This is useful technique because it allows the designer to decompose a NPC's behaviour down into manageable chunks (states) which will have their own set of unique instructions or further states (as found in a hierarchical state machine). This technique was implemented in the incredibly successful arcade game Pacman where the NPC's behaviour used a finite state machine.

However the argument stands that if the original Space Invaders did not use a FSM then why should I? This is a valid question and if I were to create a complete clone of the original game I would agree that there is no need to give the NPC's states as they are always doing more or less the same thing. However as I plan to focus on making the NPC's "intelligent" I believe it would be necessary to use a FSM approach as it will allow for multiple functionality.

5.2.3.1 Chase & Evade Example

In order illustrate the practicality of using a finite state machine with regards to this project, I came across an extremely useful article that discusses and implements two NPC's that chase and evade a player using FSM's.

As with the Galaxian game it was shown that the enemies would break away from the main group and swoop down and intercept (collide) with the player or shoot them down. The Chase & Evade example describes in detail an interesting solution to providing a similar behaviour.

The example article and source code makes use of 3 different actors, namely a cat, mouse and a tank. The tank and mouse are controlled by AI and the cat is controlled by the player. The aim of the

example is that the tank chases the player, the player chases the cat whilst evading the tank and the mouse evades the cat. However with relevance to the project my main emphasis is on the Tank NPC's behaviour as it constantly follows the player and attempts to collide with them within reason. By this I mean the tank cannot simply match the player's movements exactly and it must be bound by constraints such as speed and turning speed, otherwise it would be almost impossible for the player to not be caught.

The article describes the use of "states" to determine what the NPC should be doing given a certain circumstance. The four states are; chase, evade, caught and wander. The tank will "chase" the player if within a specific range otherwise it will wander near and around the centre of the screen. If the tank collides with the player then its state changed to "caught".

Therefore this example provides a working solution that makes use of AI states and this could be a highly suitable approach to my project. However this approach is not without setbacks, namely where the AI finds itself in an infinite loop between states. The article uses the example [5]:

"If health is less than 0.5 then
    Run away
Otherwise
    Attack"

This can result in the enemy deciding to attack during one frame but then deciding to run away in the next and so on. The ending result would be the non-player character spinning around in a circle. Another example would be where the player holds a joystick between the threshold required to run or walk. Thereby a tiny change in the joystick position will result in the player consistently and unrealistically changing between walking and running. This problem is known as "Hysteresis" and is a common occurrence in game programming.

A way around this problem is to adopt the following principle, "making a Boolean decision by testing a floating point value"[5]. Essentially this means adding a "leniency" value to the decision statement that should counteract constant flickering between states. A basic pseudo code example is as follows [5]:

"Declare a constant floating point number hysteresis = 0.1
If a value is greater than set threshold + hysteresis Then
    DoSomething();
Otherwise if value is less than set threshold – hysteresis then
    DoSomethingElse();
Otherwise
    KeepDoingWhateverYouAlreadyAre();"

5.2.4 Probabilistic Transitional Finite State Machine

After conducting the above research and following discussion I then looked at how the enemies in the game could "decide" for themselves what to do after looking at what has/is happening in the game. Essentially at this early stage my intent is for each enemy to be independently "thinking" what can I do and what should I do throughout the game. It therefore occurred to me that this would make the implementation of a finite state machine challenging due to the enemy only being allowed in one state at a time as I did not want to create a perplexing amount of states that could compromise the programmes simplicity and performance.

Therefore the concept of deciding which state to transition too based on a probability appealed to

me as it allowed for each enemy to "think" for themselves. For example an enemy may be in its idle state, casually moving with the main pack from one side of the screen to the other. However this enemy including its neighbour are eligible to break away from the main pack and fly directly towards the player in a frontal assault. Each enemy will probabilistically decide for themselves based on the same data set whether or not to do this and transition to the state at any given time. As discussed previously Bayesian Networks can provide a different reaction to a set of actions. This will give the desired effect of not repeating the same outcome (thereby making enemies unpredictable) as sometimes both enemies will break away and attack together and sometimes one will go it alone.

Based on the previous discussion regarding probability engines this element of the game AI should in theory, tie in nicely with a finite state approach using a Bayesian Network.

5.2.5 Fuzzy Logic

Fuzzy logic is another technique found in AI and is used to allow NPC's to make logical decisions based on vague instructions. For example the instructions to make cheese on toast can be given to a human like so:

- "Cut two slices of break **medium thick**
- Turn the heat on the griddle on **high**
- Grill the slices on one side until **golden brown**
- Turn the Slices over and add a **generous helping** of cheese
- Replace and grill until the top of the cheese is **slightly brown**
- Remove, sprinkle on a **small amount** of black pepper, and eat" [3]

However it is difficult to codify this so that a computer can understand such vague instructions. In terms of applying this to my project it can be useful for determining the difficulty of the AI. For example the statement, "the higher the level the player is on, the difficult the enemies should be" would be useful in its codified form as it can govern the increase in difficulty of the AI as the game progresses. As noted in the cheese on toast example, here the problem is getting the computer to understand what constitutes as a "higher level" and how rapidly should the AI increase their difficulty. This is important as the player does not want to be able to breeze their way to round 9 and then all of sudden get instantly killed on round 10 because the enemies have suddenly decided this constitutes as a high level and should make it near impossible to complete.

Another promising example for my project would be to add different difficulty settings in which the use of fuzzy logic would be even more important. This is because if a player selects the easiest difficulty it doesn't mean they want to play an easy game for several hours and attain round 1000 with ease. Fuzzy logic can be used to determine exactly how difficult the enemies should be given the current level the player is on and what difficulty has been set. Using this approach should show a distinct difference in enemy behaviour between rounds and how quickly the difficulty increases.

Therefore it may be necessary to implement a fuzzy logic element to the game AI to allow the enemies to gradually increase in difficulty over time.

5.2.6 Rule Based AI

Rule based AI, otherwise known as rule based systems is a very similar concept to that of an expert system. As implied in the name rule based AI is made up of two main components namely; an inference engine and a set of rules. This allows for NPC's to pick a certain reaction from a set list of

behaviours based upon a result provided by the inference engine. An inference engine can be regarded as a program that attempts to provide an answer from a knowledge base.

With reference to this project an inference engine can be regarded as the Bayesian Network. Therefore I can add another element to the enemies thinking arsenal such that they could decide as a group to attack “as one” in a pre programmed attack against an unwitting player. An example of this would be where each enemy fires in succession after a brief lapse in time causing the player to flee for cover or away from the incoming fire. This could also contribute toward achieving the goal of collaborative behaviour between enemies despite it only being a pre programmed response. This would therefore be classified as a game AI element as it would merely be an illusion that the enemies have consorted to act as one and this would heavily depend on the amount of different group attacks that are available. Otherwise the human player can quickly learn to anticipate these actions and act in a pre-emptive way to prevent them coming to harm.

This approach does show some promise for expansion however if the enemies are able to “choose” whether or not to take part in a group attack. Thus resulting in a more unpredictable opponent for the player in an effort to make the game more challenging and enjoyable.

5.3 Collision Detection

It is has become increasingly crucial in video games that collision detection is not only accurate but quick and efficient. This is to ensure that when something is destroyed by a laser beam then it explodes almost immediately as opposed to carrying on as if nothing has happened for a few seconds before suddenly exploding.

Due to my project solution being 2 Dimensional I anticipate the collision detection to be relatively simple due to only having 2 axes (x & y) to compare. There are several techniques available in this regard ranging from box collision to pixel collision detection. The main difference between the two is accuracy and performance. Box collision detection is incredibly simple and efficient and involves placing a rectangular "box" around each drawn graphic (also known as a sprite) and compares the bounding box coordinates for any intersections which would indicate a collision. Pixel collision is basically an extended version of box collision and uses it to detect any collisions before undertaking a pixel by pixel collision detection. Therefore after a collision has been detected using the box method this initiates a loop that runs through every pixel in the bounding box and checks for specific changes in colour. This kind of collision detection is extremely accurate at the expense of computation and performance on complex graphics such as high resolution 3 Dimensional meshes.

After doing some research in collision detection with Visual Studio it came to my attention that XNA Game Studio provides an incredibly useful Intersect function which implements box collision detection. The function works by comparing two rectangles (bounding boxes) and returning true if an intersection exists and false if there isn't. Therefore as my project will be created in 2 Dimensions where collision detection is relatively simple and was used in the original game, it is logical to make use of the Intersect function so I can focus on other aspects of the game. However if this project solution were to be in 3 dimensions it would be of high importance to plan and discuss collision detection in detail.

5.4 Power Ups & Boss Fights

As a result of my research into existing versions of the game I found that some games implemented a power up aspect to the game. This encouraged an interesting new element to the game as it would often affect the style of the player. For example a power up could be to have a rapid firing laser canon and therefore the player could adopt a much more aggressive playing style. As the main

weapon in the original Space Invaders never changes it can cause the player to lose interest fairly quickly and therefore I believe the user of power ups would remedy this issue.

The use of collision detection would be necessary in this instance as when the player collides with a power up the game must realise not to blow up the player ship (as power ups do no damage), what power up it was and what weapon does the player get as a result. In practice power ups would be randomly "dropped" by a defeated enemy and will fall at a set speed down the screen allowing the player to shoot or collide with it. As a result the player would gain the benefit of that particular power up for a limited amount of time before being returned to its normal default set up.

Although I do not believe this idea would be classed as an advanced feature from an academic point of view I do think it would add something exciting and new to a classic game and could have a profound effect on the overall experience of the game.

Another interesting element of game play is the dreaded boss fight encounters that are a frequent occurrence in almost all video games today. This can tie in nicely with the game AI for the project as the boss can be shown to display a “higher” level of intelligence and create a difficult challenge for the player to overcome.

Unfortunately however a lot of bosses tend to follow distinct patterns of behaviour (i.e. will shoot three lasers at once all of the time) and can be detrimental to the overall gaming experience. Therefore this idea can be expanded to incorporate unpredictability or intelligent behaviour as discussed in the game AI section of the report.

5.5 Networking (Multiplayer)

Traditionally Space Invaders is a single player game however making it suitable for two or more players could certainly make things more interesting to the player as it will introduce a new aspect to the game. Potentially there are two very different routes available with a multiplayer version of the game as the second player could either be their ally and aid the other player in defeating waves of enemies and compete for the high score. This idea is definitely feasible as XNA Game Studio supports the use of user hosted online sessions (client-server architecture) or a peer to peer session.

Alternatively the second player could act as the enemy commander and manipulate the enemies to their bidding in an effort to destroy the other player. As good as the idea sounds it could demand a lot of effort from an interface point of view, as the commanding of enemies can rely on the re-use of existing functions. Therefore this idea is less probable than the first but is still a feasible concept nonetheless given there is enough time remaining to design and implement it.

5.6 User Interface

The design of the user interface will be of significant importance for this project due to it being developed to on multiple platforms namely Windows OS PC’s, Windows OS Mobile Phone and Xbox 360 games console. In particular there will be a distinct visual difference between the user interface used for the Windows Phone version of the game in comparison the PC and Xbox 360 version.

This is mainly due to the hardware that will be used to run the game is vastly different when looking at a hand held device with a small screen to a computer or a games console and will therefore have different layout requirements in order to make the game interface easy to use and self-intuitive. There is also a large difference between the input devices used on each platform but most do not directly affect the user interface design. However most modern mobile phones,

especially those running the Windows Mobile OS have touch sensitive screens and allow for touch input to be used in games. This places a further requirement on interface design for the mobile version of the game to be touch input friendly. Furthermore it also requires that the game be designed to work specifically with touch input for the mobile version of the game which can be a challenge in itself.

6. Specification

From the research undertaken it is clear that this project has one aim which contains two key elements. Firstly is to create a working "clone" of the original Space Invaders game which is compatible with multiple hardware platforms. This is crucial as it will act as the backbone of the project if you will to allow for modifications and experiments to improve it. Therefore this in itself will have its own unique set of requirements/specifications which can be taken from the analysis and discussion of the original game. As a result the requirements for the cloned game are as follows:

- Player
  - Move horizontally at the bottom of the screen
  - Fire Lasers
  - Have a finite number of lives
  - Have a score associated with the number and type of enemy destroyed
- Enemies
  - Move as a group from one side of the screen to the other
  - Have a set score associated with each enemy
  - Only enemies who have the line of sight of the player can fire lasers
- Barriers
  - Absorb laser damage from both the player and the enemies
  - Have a finite amount of damage it can take before being destroyed
- Interface
  - Be intuitive and effectively display all critical information (health & score)

The second element of the project is to add "advanced features" in the ways discussed in this report. The difference here however is that not all of the objectives/requirements will be achievable for reasons such as time frame or if the modification simply does not improve on the original game considerably. With this in mind I have set the following reasonable objectives for the modification of the game solution from the first part of the project:

- Enemy AI
  - Collaborative behaviour
  - Breaking away
  - Make decisions individually based on probabilistic evidence
  - Boss fights
- User Interface
  - Design for handheld version will need to be different from console version due to screen size and input type.
- Multiplayer
  - Using one of the methodologies discussed in the networking section, create a game mode that allows more than one person to play the game at the same time.
- Power Ups
  - To allow for changes in game play and tactics.

7. Design

7.1 Approach

The approach to the design and implementation of this project will be done in incremental steps to achieve project milestones. Therefore, from the title of the project the two major project milestones are:

1. Core game engine
2. Advanced features

To achieve the first major milestone I will rely on the use of class diagrams and UML diagrams to map out what objects (or actors in UML) do I need, what do they interact with and how? From this I can then take an organised approach to the implementation of the game engine and produce a playable clone version of the game.

Of course the second major obstacle will be learning the C# programming language whilst implementing the game engine. I do not foresee this being too much of a major obstacle as there are very many resources and tutorials available.

Once the core game engine is completed I will then attempt to implement some of the discussed advanced features (AI, networking and UI). If completed I will evaluate the success of adding such a feature has on the game as if it is deemed detrimental then it should be left out.

7.2 Core Components

The core components or classes required for the core game engine are easily identifiable and it is also relatively simple to map their interactions between themselves and the game screen. If we analyse the original Space Invaders we can easily see that the main classes (or actors) required are the:

- Player
- Enemies
- Lasers
- Enemy Lasers
- Barriers
- Menu

With these fundamental classes it is also easy to classify what high level attributes are associated with them. For example it is obvious that both the player, enemies and barriers will have some sort of health attribute. We ignore exactly how the health for each of these classes works at this stage and focus on just the attributes. Another less obvious attribute (despite it staring you in the face) is the position of every graphic on the screen. Therefore all of the above named classes will require a position attribute so the game engine knows where everything is to draw it.

Finally each of the actors that is required to move will need to have a speed attribute defined to limit how quickly they move about in 2 Dimensional space. Therefore with this simple “say what you see” approach to defining classes and attributes I have already set the basis of the class design which can then be progressed to discuss their methods.

8. Conclusion

From this report it has been concluded that Visual Studio 2010 IDE is the most suitable environment for creating a multiplatform game due to its vast resources. It is also apparent from detailed analysis of the original Space Invaders and modern adaptations of the original game, all share similar flaws such as AI, in which can be improved on. As discussed there are many well documented techniques such as FSM's and Bayesian networks that can be used as a framework for decision making which can be perceived by the user as intelligent.

Furthermore it is evident that there are other areas in which the original game can be improved, such as graphics, interface and game play (as shown by Space Invader 5 and Galaxian). Also it has been concluded that adapting the original features of the game for a hand held device will prove challenging in itself with regards to designing an efficient user interface for touch input.

Whereas it was initially planned to go through the design of the project in detail at this stage it had become evident that there was a plethora of background research required before moving onto design. This can therefore be expected in the final report as well as detailed implementation, testing, evaluation and overall conclusion.

Therefore in conclusion it has been shown the Space Invaders is a great game in which to recreate and adapt for modern platforms which justifies this projects current direction and overall aim.

8.1 Further Work

The next steps in this project and for the final report are to continue with the design section in a more detail and thorough manner. Also the implementation of the core game engine will be completed and optimized to allow for easy modification to begin testing advanced features. This will be done in the form of a project diary in which I will describe the step by step progress of implementation and discuss the challenges faced along the way. After this I will begin the alpha testing phase of the project (note that testing is done iteratively during the implementation by myself to test specific things) in which I will undertake a detailed analysis and bug report before releasing the game for public (beta) testing.

Finally from this I will draw up an evaluation of the project on the whole and discuss its outcomes. Moreover I will compare and contrast the solution produced compared to the concept initially thought up during the initial plan phase of the project and from this I will state a conclusion with regards to the overall success of the project.

9. Appendices

9.1 Abbreviations

- CPU
  - Central Processing Unit
- GPU
  - Graphics Processing Unit found in graphics cards designed for processing graphical calculations to display on a screen
- AI
  - Artificial Intelligence
- NPC
  - Non Player Character that is controlled by some form of artificial intelligence
- OS
  - Operating System
- IDE
  - Integrated Development Environment
- UI
  - User Interface

9.2 References

1. Buckland, M. (2004). Programming Game AI by Example. ISBN 1-55622-078-2
2. Chasing and Evading Example, 2010 [Online]Available at: < http://create.msdn.com/en-US/education/catalog/sample/chase_evade>[Accessed 12 December 2011]
3. Cummings, J and Elzer, S and Zoppetti, G. (2007). Bayesian Networks in Video Games [Online] Available at: <http://cs.millersville.edu/lib/userfiles/selzer/publications/Cummings-final.doc> [Accessed 12 December 2011]
4. Meisner, E. (2003). Naive Bayes Classifier Example [Online] Available at: <http://www.inf.u-szeged.hu/~ormandi/ai2/06-naiveBayes-example.pdf> [Accessed 12 December 2011>
5. Mitchell, T. (1997). Machine Learning, McGraw Hill. ISBN 0-07-042807-7, p.2