**Part 4 – Final Project Submission, Demo & Presentation (60 marks)**

**Final Project Deliverables**

The deadline for the final project submission is scheduled on **8th October 2020 (Thu)** of **Week 14** at 0959 hours. Project presentation and demo will be held on the **8th October 2020 (Thu)** of **Week 14** during the usual lab session.

Please submit a **ZIP archive**, named with your **game title**, which contains the following softcopies to MMLS:

**1) Final Report**

Each project group must hand in **ONE** final report. Your final report file should be named with your **game title** and **both student IDs**, and it should include the following:

[+] **An introduction and overview of your game**

[+] **Game design** (can include character/level/layout designs)

[+] **Technical design** (detailed explanation on core techniques and game AI algorithms

implemented in your game)

[+] **Game development issues** (see list below)

[+] A brief **user manual** on how to play the game

Your final report should address the following game development issues:

[+] What is the coolest thing about your game (attraction point or selling point)

[+] What changes you made to your original game design plan for

[-] technical reasons and why

[-] playability reasons and why

[+] What you would do next if you had more time

[+] What you would do differently next time.

Please include screenshots of your game, and other important figures, diagrams and charts.

**2) Presentation & Game Demo**

Please prepare presentation slides (preferably in **.ppt, .pptx**, or **.pdf** formats) for your project presentation. Both students in each project are expected to present (hence, make your own arrangements). There is no “format” for the presentation. You are free to decide on the content that best showcases your work and accomplishments.

Presentation and game demo will be held during the lab session in Week 14 (since this has been fixed early on, please ensure that your other presentations do not clash with this). Please limit the duration of your presentation to a maximum of 10 minutes, including a quick demo of your game. In the short Q&A session after that, you may be directed some questions and comments. You are encouraged to answer the questions posed to you honestly and professionally.

**3) Program Folder**

The Program Folder should contain **complete project source codes, assets, other resource files, etc.** You may also include other miscellaneous materials that you have compiled, used, or referred to.

Contents

[1.0 Introduction and Overview 1](#_Toc52920692)

[1.1 Win Condition 1](#_Toc52920693)

[1.2 Lose Condition 1](#_Toc52920694)

[1.3 Challenges 1](#_Toc52920695)

[1.4 Rewards and Punishment 2](#_Toc52920696)

[2.0 Game Design 3](#_Toc52920697)

[2.1 Character design 3](#_Toc52920698)

[2.2 Level design 6](#_Toc52920699)

[3.0 Technical Design 7](#_Toc52920700)

[3.1 Movement 7](#_Toc52920701)

[3.2 Line of Sight 9](#_Toc52920702)

[3.3 Pathfinding 11](#_Toc52920703)

[3.4 Decision-making 12](#_Toc52920704)

[4.0 Game Development Issues 13](#_Toc52920705)

[4.1 Cool Features 13](#_Toc52920706)

[4.2 Changes to Original Plan 13](#_Toc52920707)

[4.3 Potential Changes (if we had more time) 14](#_Toc52920708)

[4.4 What we would do differently next time 14](#_Toc52920709)

[5.0 User Manual 15](#_Toc52920710)

[5.1 How to play? 15](#_Toc52920711)

[5.2 Game Input 16](#_Toc52920712)

[References: 17](#_Toc52920713)

# 1.0 Introduction and Overview

Shoot ‘em up games, also commonly known as shmup or STG, is a part of the shooting subgenre of video games in the action genre. The subgenre has multiple categories defined by its design elements, i.e. viewpoint and movement, such as rail shooters, scrolling shooter, bullet hell, and run and gun. The common elements of this subgenre are:

* Top-down perspective or side-view perspective
* Player must use ranged weapons to perform action
* Player’s avatar is usually a vehicle that is constantly under attack
* Player’s goal is to shoot at anything that moves and destroy them
* Player may endure some damage before destruction
* Player need fast reaction and some form of memorisation of the enemy’s pattern

Our game is a shoot ‘em up game with a top-down perspective with a sci-fi and space theme.

## 1.1 Win Condition

The player needs to progress through the stage without getting destroyed by the barrage of bullets from the enemies and defeat the boss at the end of the stage.

## 1.2 Lose Condition

The player loses all their health bars.

## 1.3 Challenges

* Turrets are placed at specific positions. Turrets will attack the player once the player is within sight. Turrets can be destroyed after 10 hits and their output damage is 1 health bar per bullet.
* Asteroids will wander around on the screen. If the asteroid clashes with the player, the asteroid will be destroyed, and the player loses 1 health bar. Players can destroy the asteroid by firing bullets, which deal 1 damage each.
* Enemy Type 1 will move in a group. Enemy will attack the player once the player is within sight. Each enemy has 5 health bars. If the enemy clashes with the player, each party will lose 1 health bar.
* Enemy Type 2 will move towards the target. Enemy will attack the player once the player is within sight. Each enemy has 7 health bars. If the enemy clashes with the player, each party will lose 1 health bar.
* Boss is controlled by the decision-making AI, this means the boss will dodge the bullet from the player, attack the player at a suitable time and become stronger when the health bar reaches a certain amount. Boss has 50 health bars. If the boss clashes with the player, each party will lose 1 health bar.

## 1.4 Rewards and Punishment

Rewards: There will be life pickup for the player to collect if the player can survive for a certain distance. The score will increase after the enemy is killed. Different enemies have different scores.

Punishment: If the player dies, they have to restart and start from the beginning of the level again.

# 2.0 Game Design

## 2.1 Character design

### 2.1.1 Player’s avatar

The player can move the player’s avatar using the arrow keys; fire a missile using the “z” key and the selection of the target is through “Tab” key; shoot bullets using the “spacebar” key.

### 2.1.2 Asteroid

We want the asteroids to behave similarly to its real-life counterpart. The asteroids will wander around randomly throughout the screen through the dynamic wander algorithm. When it collides against the player, it causes damage.

### 2.1.3 Enemy1 (small enemy ship)

The small enemy ship will move towards the player through the kinematic seek algorithm. The small enemy ship will fire at the player once the distance reaches a certain limit and is within the angular distance (cone) of the player. This process is done through the line-of-sight algorithm. Other than shooting at the player, the small enemy ship will try to collide with the player too.

### 2.1.4 Enemy2 (big enemy ship)

### 2.1.5 Turret

Just like the name suggested, it is a stationary enemy. It shoots at the player if the line of sight is clear.

Our character design can be summarized with the table below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Game Object | Health bar | Destroyable | Can shoot? | Bullet damage | Clash damage | Movement | Line of sight | Decision-making |
| Player’s avatar | 5 | ✔️ | ✔️ (bullet, missile) | 1 | 1 | Player controlled | Player’s | ❌ |
| Missile | ❌ | ❌ | ❌ | 3 | 3 | Pathfinding | ❌ | ❌ |
| Asteroid | 1 | ✔️ | ❌ | ❌ | 1 | Kinematic wander | ❌ | ❌ |
| Turret | 10 | ✔️ | ✔️ | 1 | 1 | Stationary | ✔️ | ✔️ |
| Enemy1 | 5 | ✔️ | ✔️ | 1 | 1 | Pattern movement | ✔️ | ❌ |
| Enemy2 | 7 | ✔️ | ✔️ | 1 | 1 | Kinematic seek | ✔️ | ❌ |
| Boss | 50 | ✔️ | ✔️ | 1 | 1 | Stationary, pattern movement | ✔️ | ✔️ |

Table 2-1: Properties of Game Objects

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Texture** | **Class Name** | **Technical aspects** | | | |
| **Health** | **Movement Speed** | **Fire rate** | **Bullet Texture** |
|  | Player | 5 | 300.0f | 150.0f | (Normal bullet)    (Missile) |
|  | Asteroid | 1 | 100.0f | - | - |
|  | Turret |  | - |  |  |
|  | Enemy1 |  | 100.0f | 500.0f |  |
|  | Enemy2 |  | 100.0f | 100.0f |  |
|  | Boss | 50 | 100.0f | 150.0f |  |

Table 2-2: Technical Details of Game Objects

## 2.2 Level design

# 3.0 Technical Design

## 3.1 Movement

* Kinematic Seek

Kinematic seek allows the game object to move towards a target. This movement algorithm is the same as the one discussed in class.

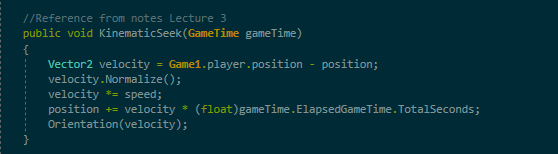


Figure 3-1: Code implementation of Kinematic Seek

* Dynamic Wander

This movement algorithm allows objects to move randomly in different directions, thus changing the object’s orientation.

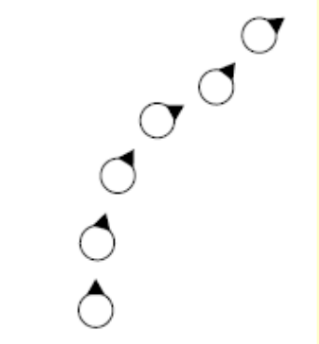


Figure 3-2: Movement of Dynamic Wander

According to Pandey, as compared to kinematic wandering, a few more parameters are needed for dynamic wandering, including max speed, max acceleration, wander radius, wander offset, and wander rate in order to generate a better and smoother movement. A wander point is set in a circle of wander radius, which is wander offset units forwards from the player. Wader rate refers to the frequency of orientation change. The object will align itself with the target and moves towards the direction of the target. Once the object reaches the boundaries (goes off screen), the object will be destroyed.

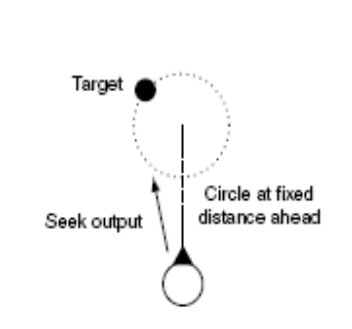


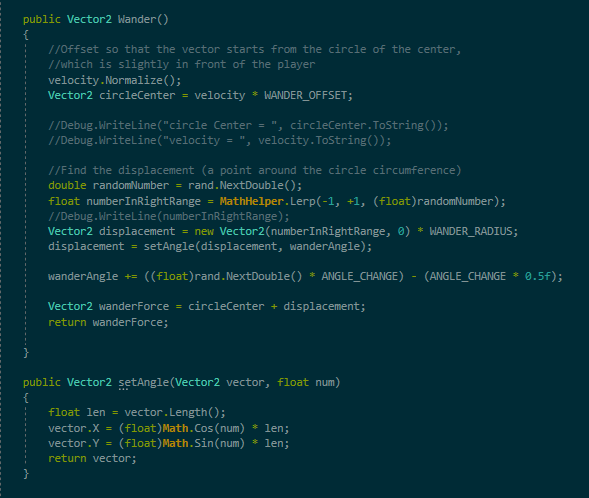
Figure 3-3: Dynamic Wander AI 

Figure 3-4: Code implementation of Dynamic Wander

* Pattern Movement

This movement algorithm allows objects to move in an organized manner. We plan to employ the classic Catmull-Rom spline curve to interpolate a smooth path between a few points, so that the game objects can move across the screen in a curved line.

Figure 2-6: Code implementation of Catmull-Rom spline movement

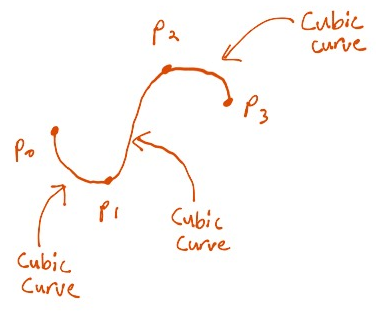


Figure 3-5: Catmull-Rom spline curve

## 3.2 Line of Sight

Line of sight refers to what the game objects can see and how they should react to it. Bresenham’s line scan algorithm draws an approximated straight line on the screen between two endpoints. Objects can exhibit a natural behaviour through visual limit, free sight, and Bresenham’s algorithm, where lines are drawn between two objects to check whether they can see each other or not.

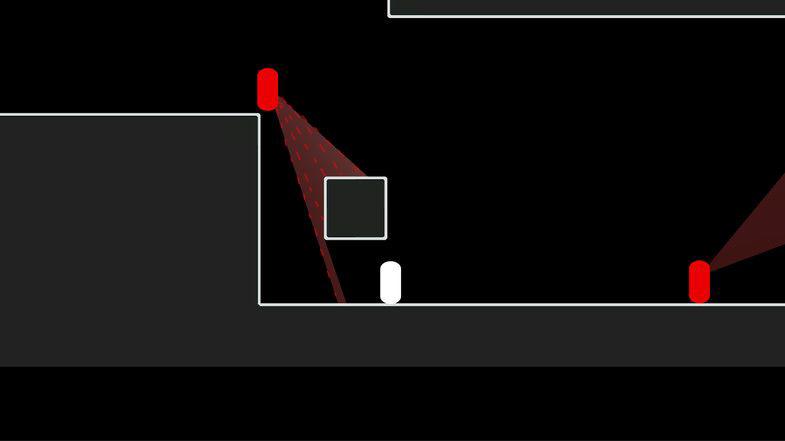


Figure 3-7: Line of sight (cone-shaped)

Visual limit is the limitation of the distance of their sight, e.g. half of the window size. Free sight refers to an uncollided or unblocked line between the two objects. An easier way compared to drawing lines is to check whether the object is within the radius of another object through a simple calculation of their distance apart. However, this does not take into account whether there are other objects blocking their line of sight.

For our line of sight calculation, we first check the distance by comparing the two game objects’ positions and the angle between them. If both the distance and angle are within the specified limits, we then do a Bresenham line check. For each point generated from the Bresenham line check, we check whether the enemy’s bounding box in the enemy list contains the point or not. If it contains such point, then it means the line of sight is blocked by another enemy. Otherwise, the line of sight is clear.

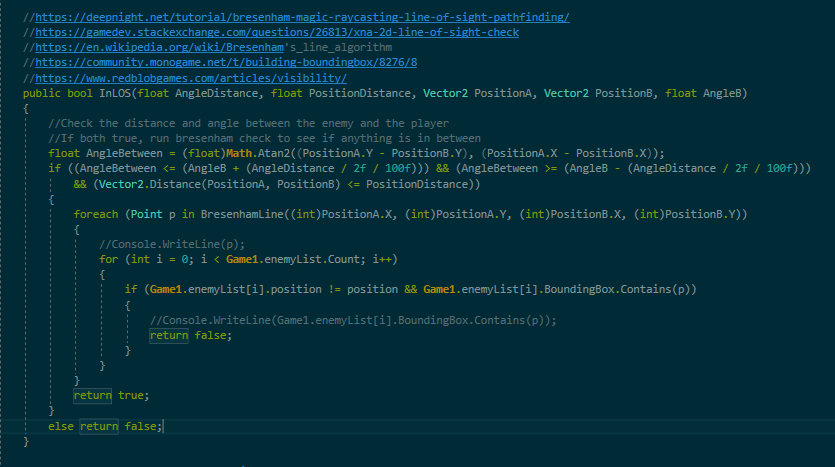


Figure 3-8: Code implementation of line of sight check

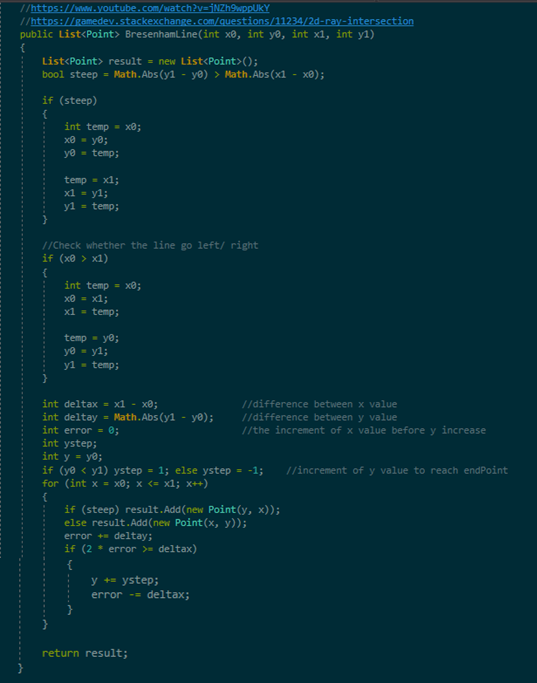


Figure 3-9: Code implementation of Brensenham’s line check

## 3.3 Pathfinding

Pathfinding allows the game objects to move to another point using the shortest path. The common algorithm for pathfinding is Dijkstra’s algorithm and A\* algorithm. For this game, we plan to use A\* algorithm for pathfinding. We choose to use A\* algorithm is because this algorithm is optimal.



Figure 3-10: A\* algorithm

## 3.4 Decision-making

Decision-making algorithms allow game objects to change their behavior when there is a change of circumstances. For example, the ghosts in Pac-Man change their behavior from seeking to fleeing, once Pac-Man consumes a super pill. Finite state machine is one of the oldest forms of game AI, but it is simple and it produces a great result when it comes to object behaviour.

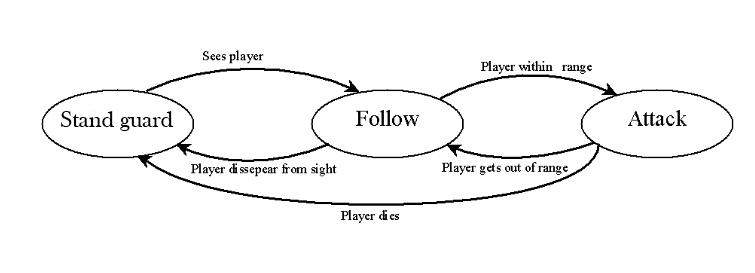


Figure 3-11: A simple finite state machine for boss

# 4.0 Game Development Issues

Your final report should address the following game development issues:

[+] What is the coolest thing about your game (attraction point or selling point)

[+] What changes you made to your original game design plan for

[-] technical reasons and why

[-] playability reasons and why

[+] What you would do next if you had more time

[+] What you would do differently next time.

## 4.1 Cool Features

## 4.2 Changes to Original Plan

We have made several changes to the original plan to enhance its playability. We added the following features:

**1. Powerup**

**2. Tutorial (pop-up)**

**3. Pause Scene**

When player presses the “Enter” key, the game comes to a stop, and the player can choose to continue or quit the game.

Other than playability reasons, we also made these following changes to our original game design plan due to technical issues:

**1. Pathfinding**

We originally planned to use A\* algorithm for the missile tracking. Since our game employs continuous environment, where each coordinate is real number, the A\* algorithm takes too long to execute and the game stutters temporarily. To solve the issue, we combined the Bresenham’s line check used in line-of-sight together with A\* algorithm.

## 4.3 Potential Changes (if we had more time)

## 4.4 What we would do differently next time

# 5.0 User Manual

## 5.1 How to play?



When the player first launches our game, Space Battle, the player will see the screen above. The player will have to use their mouse to click on the START! button or EXIT button. Clicking on the START! button leads player to the gameplay, while clicking on the EXIT button will exit the game.

## 5.2 Game Input

|  |  |
| --- | --- |
| **Keys** | **In-game action** |
| ↑ | Move forward |
| ↓ | Move backward |
| ← | Move left |
| → | Move right |
| Space bar | Fire bullet |
| Z | Fire missile |
| Tab | Select enemy to fire missile |
| Enter | Pause Game |

# References:

1. <https://en.wikipedia.org/wiki/Shoot_%27em_up>
2. <http://www.codenamepandey.com/movementalgo>
3. <https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-wander--gamedev-1624>
4. <http://www.cs.uu.nl/docs/vakken/b2ki/LastYear/Docs/Slides/motion-handouts.pdf>
5. <https://andrewhungblog.wordpress.com/2017/03/03/catmull-rom-splines-in-plain-english/>
6. <https://answers.unity.com/questions/46366/how-do-i-handle-pathfinding-in-shoot-em-ups.html>
7. <https://gamedev.stackexchange.com/questions/28204/ai-algorithm-for-avoiding-bullets-in-a-shoot-em-up-game>
8. <https://gamedevelopment.tutsplus.com/tutorials/build-a-stage3d-shoot-em-up-terrain-enemy-ai-and-level-data--active-11160>
9. <https://www.diva-portal.org/smash/get/diva2:4762/FULLTEXT01.pdf>
10. <https://en.wikipedia.org/wiki/Pathfinding>
11. <https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1063&context=itbj>
12. <https://stackoverflow.com/questions/54391261/pathfinding-to-a-moving-target>
13. <https://mukulkakroo.com/2019/01/30/ai-movement-algorithms/>
14. <https://www.geeksforgeeks.org/a-search-algorithm/>