

TGD3351
GAME ALGORITHMS

Game Title:

Space Battle



FACULTY OF COMPUTING AND INFORMATICS

TRIMESTER 1 2020/2021

12 OCTOBER 2020

| | |
|--------------------------------|--------------------------------------------|
| Student ID : 1171101517 | Student Name: Michelle Chai Mei Wei |
| Student ID : 1171100973 | Student Name: Foo Fang Jee |

Contents

| | | |
|-------|-----------------------------------------------|----|
| 1.0 | Introduction and Overview | 1 |
| 1.1 | Win Condition..... | 1 |
| 1.2 | Lose Condition..... | 1 |
| 1.3 | Challenges | 1 |
| 1.4 | Rewards and Punishment | 2 |
| 2.0 | Game Design | 3 |
| 2.1 | Character design | 3 |
| 2.1.1 | Player's avatar..... | 3 |
| 2.1.2 | Asteroid..... | 3 |
| 2.1.3 | Enemy1 (small enemy ship) | 3 |
| 2.1.4 | Enemy2 (big enemy ship)..... | 3 |
| 2.1.5 | Turret | 4 |
| 2.2 | Level design | 7 |
| 3.0 | Technical Design | 9 |
| 3.1 | Movement..... | 9 |
| 3.2 | Line of Sight..... | 12 |
| 3.3 | Pathfinding | 14 |
| 3.4 | Decision-making | 16 |
| 4.0 | Game Development Issues | 19 |
| 4.1 | Cool Features..... | 19 |
| 4.2 | Changes to Original Plan | 19 |
| 4.3 | Potential Changes (if we had more time) | 20 |
| 5.0 | User Manual..... | 21 |
| 5.1 | How to play?..... | 21 |
| 5.1.1 | Main Menu..... | 21 |
| 5.1.3 | Tutorial..... | 22 |
| 5.1.3 | Gameplay HUD..... | 23 |
| 5.1.3 | Game Input | 24 |
| 5.1.4 | Pause Game | 25 |
| 5.1.5 | Game Over | 26 |
| 6.0 | References | 27 |

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 towards the target. 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 in a group. 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. The boss at level 1 has 25 health bars. The boss at level 2 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. The player's avatar has 10 health bars and a missile cooldown of 5 seconds.

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)

The big enemy ship will move towards the player through a pattern movement algorithm. The pattern movement algorithm employs the Catmull-spline curve calculations to find the positions along four control points. The enemies will move along these calculated positions as if they are waypoints. Just like the small enemy ship, the big enemy ship will fire at the player once the distance reaches a certain limit and is within the angular distance (cone) of the player (line-of-sight algorithm). The collision of big enemy ship and player will cause damage too.

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.

2.1.6 Boss

There are two bosses in the game (level 1 and level 2). Both bosses are controlled by the finite state machine. This means the bosses will enter an avoid state and dodge the bullet when the player is firing. However, the bosses will only avoid for 1 second and after that, they will enter an attack state. If the player is still firing, the state will change from attack to avoid again after 2 seconds. The same rule applies on the attack faster state, but the 2 seconds delay is reduced to 1 second. Both attack state and attack faster state will chase the player and fire towards the player. The bosses will enter attack faster state after the health reaches 30% of maximum health. The firing rate for both bosses are different as explained below:

2.1.6 (a) Boss (Level 1)

The attack state and attack faster state are different in terms of firing rate, in which the attack state will fire at rate of 250 milliseconds while the attack faster state will fire at rate of 160 milliseconds.

2.1.6 (b) Boss (Level 2)

The attack state and attack faster state are different in terms of firing rate, in which the attack state will fire at rate of 200 milliseconds while the attack faster state will fire at rate of 160 milliseconds

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 | 10 | ✓ | ✓ (bullet, missile) | 1 | 1 | Player controlled | Player's | ✗ |
| Missile | ✗ | ✗ | ✗ | 3 | 3 | Pathfinding | ✗ | ✗ |
| Asteroid | 1 | ✓ | ✗ | ✗ | 1 | Dynamic wander | ✗ | ✗ |
| Turret | 10 | ✓ | ✓ | 1 | 1 | Stationary | ✓ | ✓ |
| Enemy1 | 3 | ✓ | ✓ | 1 | 1 | Kinematic seek | ✓ | ✗ |
| Enemy2 | 7 | ✓ | ✓ | 1 | 1 | Pattern movement | ✓ | ✗ |
| Boss (Level 1) | 25 | ✓ | ✓ | 1 | 1 | Stationary | ✓ | ✓ |
| Boss (Level 2) | 50 | ✓ | ✓ | 1 | 1 | Stationary | ✓ | ✓ |

Table 2-1: Properties of Game Objects















| Texture | Class Name | Technical aspects | | | |
|-------------------------------------------------------------------------------------|----------------|-------------------|-------------------|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | | Health | Movement Speed | Fire rate | Bullet Texture |
|  | Player | 5 | 300.0f | 300.0f |  (Normal bullet)  (Missile) |
|  | Asteroid | 1 | 100.0f | - | - |
|  | Turret | 10 | - | 600.0f |  |
|  | Enemy1 | 3 | 100.0f | 700.0f |  |
|  | Enemy2 | 7 | 150.0f | 300.0f |  |
|  | Boss (Level 1) | 25 | 200.0f | 250.0f, 160.0f |  |
|  | Boss (Level 2) | 50 | 200.0f | 200.0f, 160.0f |  |

Table 2-2: Technical Details of Game Objects

2.2 Level design

There will be 2 levels in this game. The game consists of 4 types of normal enemy, which are asteroid, turret, enemy1, and enemy2 and 2 types of boss. Asteroid and enemy1 will come out after every 2 seconds while turret will come out every 10 seconds. Asteroid will come out from the top and wander around the map. Enemy1 will come out from top left and seek for the player. Turret will spawn at fixed position. Since the player is moving upward, the turret will slowly move down. The same process will continue until the boss is comes out.

At a distance of 2,000, a boss will spawn out at the top centre. At the same time, all the enemies on the screen will be cleared, and no more enemies will be spawned when the boss is alive. In level 1, the boss has 25 health bars. A powerup that changes the bullet pattern of the player will be dropped after the completion of level 1. Player can proceed to level 2 after the boss is died. The same process will be applied to level 2, but the health bars of the boss is increased to 50. Player will win the game after killing the boss in level 2.

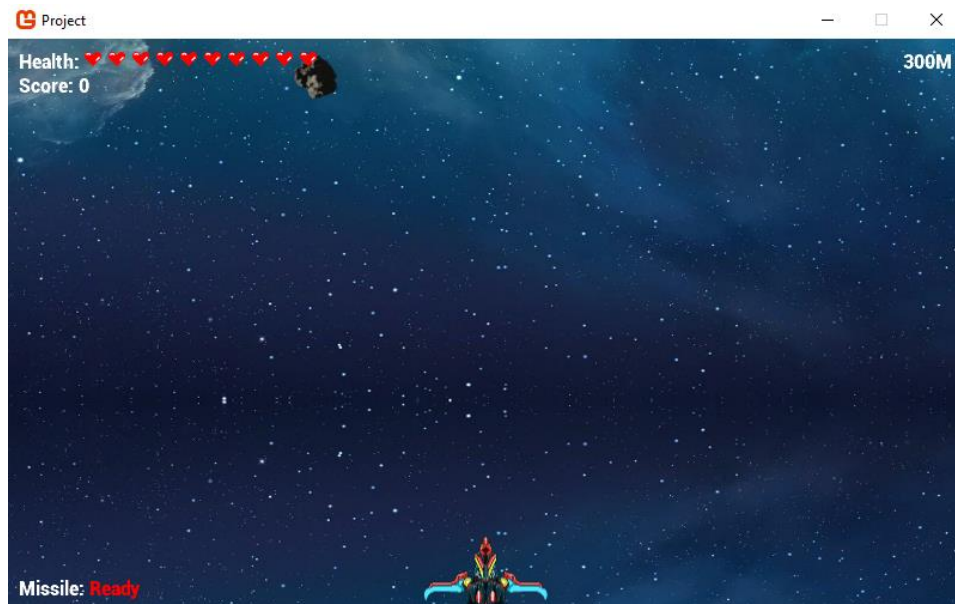


Figure 2-1: Level 1



Figure 2-2: Level 2

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.

```
//Reference from notes Lecture 3
public void KinematicSeek(GameTime gameTime)
{
    Vector2 velocity = Game1.player.position - position;
    velocity.Normalize();
    velocity *= speed;
    position += velocity * (float)gameTime.ElapsedGameTime.TotalSeconds;
    Orientation(velocity);
}
```

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. Wander 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.

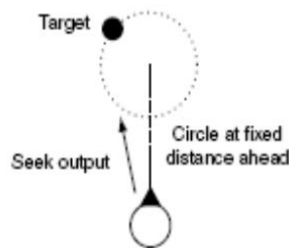


Figure 3-3: Dynamic Wander AI

```
public Vector2 Wander()
{
    //Offset so that the vector starts from the circle of the center,
    //which is slightly in front of the player
    velocity.Normalize();
    Vector2 circleCenter = velocity * WANDER_OFFSET;

    //Debug.WriteLine("circle Center = ", circleCenter.ToString());
    //Debug.WriteLine("velocity = ", velocity.ToString());

    //Find the displacement (a point around the circle circumference)
    double randomNumber = rand.NextDouble();
    float numberInRightRange = MathHelper.Lerp(-1, +1, (float)randomNumber);
    //Debug.WriteLine(numberInRightRange);
    Vector2 displacement = new Vector2(numberInRightRange, 0) * WANDER_RADIUS;
    displacement = setAngle(displacement, wanderAngle);

    wanderAngle += ((float)rand.NextDouble() * ANGLE_CHANGE) - (ANGLE_CHANGE * 0.5f);

    Vector2 wanderForce = circleCenter + displacement;
    return wanderForce;
}

public Vector2 setAngle(Vector2 vector, float num)
{
    float len = vector.Length();
    vector.X = (float)Math.Cos(num) * len;
    vector.Y = (float)Math.Sin(num) * len;
    return vector;
}
```

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.

```
private void PatternMovement()
{
    for (int i = 0; i < controlPointsList.Count; i++)
    {
        if (i == 0 || i == controlPointsList.Count - 1 || i == controlPointsList.Count - 2)
            continue;
        FindCatmullRomSpline(i);
    }
}

void FindCatmullRomSpline(int pos)
{
    //Four points needed to form a spline between p1 and p2
    Vector2 p0 = controlPointsList[pos - 1];
    Vector2 p1 = controlPointsList[pos];
    Vector2 p2 = controlPointsList[pos + 1];
    Vector2 p3 = controlPointsList[pos + 2];

    //Start position
    Vector2 lastPos = p1;

    //The spline's resolution (make sure it adds up to 1, 0.2+0.2+0.2+0.2 = 0.8, or 0.3
    float resolution = 0.2f;

    //How many times to loop
    int loops = (int)Math.Floor(1f / resolution);
    for (int i = 1; i <= loops; i++)
    {
        float t = i * resolution; // t position
        Vector2 newPos = GetCatmullRomPosition(t, p0, p1, p2, p3);
        positionList.Add(newPos);
        lastPos = newPos;
    }
}

Vector2 GetCatmullRomPosition(float distance, Vector2 p0, Vector2 p1, Vector2 p2, Vector2 p3)
{
    //The coefficients of the cubic polynomial
    Vector2 a = 2f * p1;
    Vector2 b = p2 - p0;
    Vector2 c = 2f * p0 - 5f * p1 + 4f * p2 - p3;
    Vector2 d = -p0 + 3f * p1 - 3f * p2 + p3;

    //The cubic polynomial: a + b * t + c * t^2 + d * t^3
    Vector2 pos = 0.5f * (a + (b * distance) + (c * distance * distance) + (d * distance * distance * distance));

    return pos;
}

public void LineOfSight(GameTime gameTime)
{
    if (gameTime.TotalGameTime.TotalMilliseconds > fireTime)
    {
        fireTime = (float)gameTime.TotalGameTime.TotalMilliseconds + fireRate;

        if (InLOS(90, 300, Game1.player.position, position, orientation))
        {
            EnemyBullet tempBullet = new EnemyBullet();
            tempBullet.setOwner(this);
            tempBullet.Initialize();
            Game1.enemyBulletList.Add(tempBullet);
        }
    }
}
```

Figure 3-5: Code implementation of Catmull-Rom spline movement

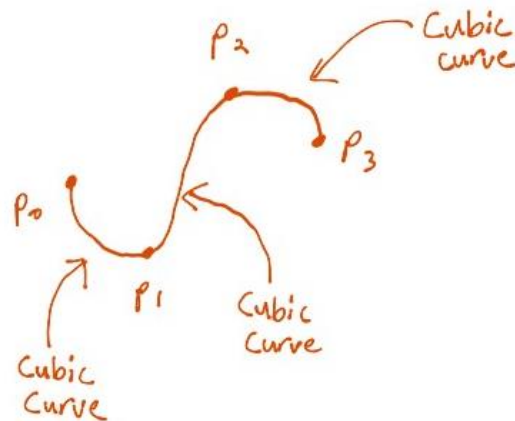


Figure 3-6: Catmull-Rom spline curve

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.

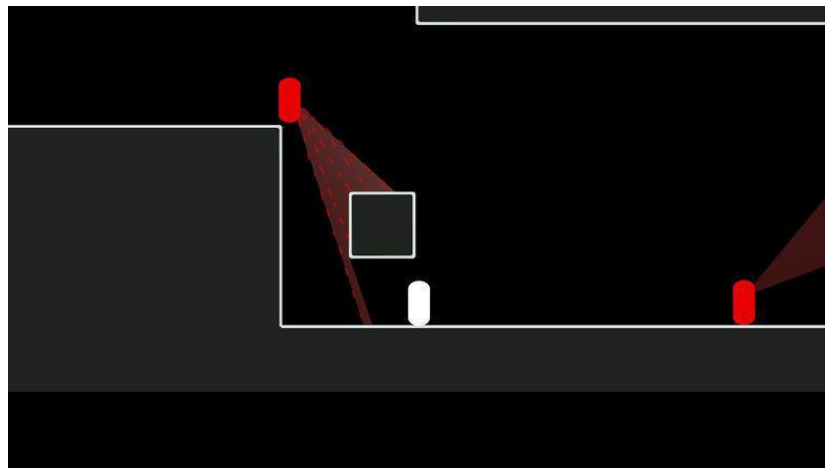


Figure 3-7: Line of sight (cone-shaped)

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.

```
//https://deepnight.net/tutorial/bresenham-magic-raycasting-line-of-sight-pathfinding/  
//https://gamedev.stackexchange.com/questions/26813/xna-2d-line-of-sight-check  
//https://en.wikipedia.org/wiki/Bresenham's_line_algorithm  
//https://community.monogame.net/t/building-boundingbox/8276/8  
//https://www.redblobgames.com/articles/visibility/  
public bool InLOS(float AngleDistance, float PositionDistance, Vector2 PositionA, Vector2 PositionB, float AngleB)  
{  
    //Check the distance and angle between the enemy and the player  
    //If both true, run bresenham check to see if anything is in between  
    float AngleBetween = (float)Math.Atan2((PositionA.Y - PositionB.Y), (PositionA.X - PositionB.X));  
    if ((AngleBetween <= (AngleB + (AngleDistance / 2f / 100f))) && (AngleBetween >= (AngleB - (AngleDistance / 2f / 100f)))  
        && (Vector2.Distance(PositionA, PositionB) <= PositionDistance))  
    {  
        foreach (Point p in BresenhamLine((int)PositionA.X, (int)PositionA.Y, (int)PositionB.X, (int)PositionB.Y))  
        {  
            //Console.WriteLine(p);  
            for (int i = 0; i < Game1.enemyList.Count; i++)  
            {  
                if (Game1.enemyList[i].position != position && Game1.enemyList[i].BoundingBox.Contains(p))  
                {  
                    //Console.WriteLine(Game1.enemyList[i].BoundingBox.Contains(p));  
                    return false;  
                }  
            }  
        }  
        return true;  
    }  
    else return false;  
}
```

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

```

//https://www.youtube.com/watch?v=iN2h9wppUkY
//https://gamedev.stackexchange.com/questions/11234/2d-ray-intersection
public List<Point> BresenhamLine(int x0, int y0, int x1, int y1)
{
    List<Point> result = new List<Point>();
    bool steep = Math.Abs(y1 - y0) > Math.Abs(x1 - x0);

    if (steep)
    {
        int temp = x0;
        x0 = y0;
        y0 = temp;

        temp = x1;
        x1 = y1;
        y1 = temp;
    }

    //Check whether the line go left/ right
    if (x0 > x1)
    {
        int temp = x0;
        x0 = x1;
        x1 = temp;

        temp = y0;
        y0 = y1;
        y1 = temp;
    }

    int deltax = x1 - x0;           //difference between x value
    int deltax = Math.Abs(y1 - y0); //difference between y value
    int error = 0;                  //the increment of x value before y increase
    int ystep;
    int y = y0;
    if (y0 < y1) ystep = 1; else ystep = -1; //increment of y value to reach endPoint
    for (int x = x0; x <= x1; x++)
    {
        if (steep) result.Add(new Point(y, x));
        else result.Add(new Point(x, y));
        error += deltax;
        if (2 * error >= deltax)
        {
            y += ystep;
            error -= deltax;
        }
    }

    return result;
}

```

Figure 3-9: Code implementation of Bresenham'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.

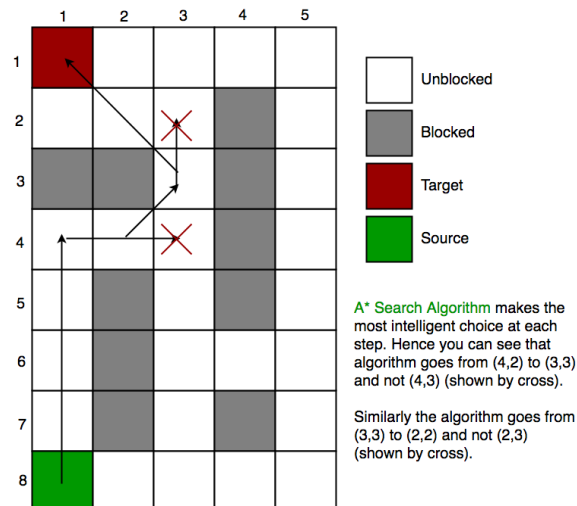


Figure 3-10: A* algorithm

```
public static List<Vector2> Compute(Vector2 start, Vector2 goal)
{
    start = new Vector2((int)start.X, (int)start.Y);
    goal = new Vector2((int)goal.X, (int)goal.Y);
    PriorityQueue<Vector2, int> priorityQueue = new PriorityQueue<Vector2, int>();
    IDictionary<Vector2, Vector2> comeFrom = new Dictionary<Vector2, Vector2>();
    IDictionary<Vector2, int> costSoFar = new Dictionary<Vector2, int>();

    if (Compute2(start, goal))
    {
        System.Diagnostics.Debug.WriteLine("bresenham");
        List<Vector2> path = new List<Vector2>();
        path.Add(goal);
        return path;
    }
    else
    {
        System.Diagnostics.Debug.WriteLine("bresenham to reduce the node");
        if (lastPoint != Vector2.Zero)
            start = lastPoint;
    }

    Initialize(start, goal);
    walkablePosition.Clear();
    WalkablePosition(start, goal);

    priorityQueue.Enqueue(start, 0);
    IEnumerable<Vector2> validNode = walkablePosition.Where(x => x.Value).Select(x => x.Key);

    foreach (Vector2 node in validNode)
        costSoFar.Add(new KeyValuePair<Vector2, int>(node, int.MaxValue));

    comeFrom[start] = start;
    costSoFar[start] = 0;
}
```

Figure 3-11: Pathfinding Implementation

```

while (priorityQueue.GetCount() > -1)
{
    Vector2 curr = priorityQueue.Dequeue();

    if (curr == goal)
        break;

    List<Vector2> neighbour = GetNeighbour(start, goal, curr, validNode);

    foreach (Vector2 node in neighbour)
    {
        int newCost = costSoFar[curr] + Heuristic(curr,goal);

        try
        {
            if (costSoFar[node] == costSoFar[goal] || newCost < costSoFar[node])
            {
                costSoFar[node] = newCost;
                int priority = newCost + Heuristic(node, goal);
                priorityQueue.Enqueue(node, priority);
                if(!comeFrom.ContainsKey(node))
                    comeFrom.Add(new KeyValuePair<Vector2, Vector2>(node, curr));
                else
                    comeFrom[node] = curr;
            }
        }
        catch
        {
            //do nothing
        }
    }
}

return ConstructPath(comeFrom, start, goal);
}

```

Figure 3-12: Pathfinding Implementation

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. Our FSM is implemented in the boss class and our game class.

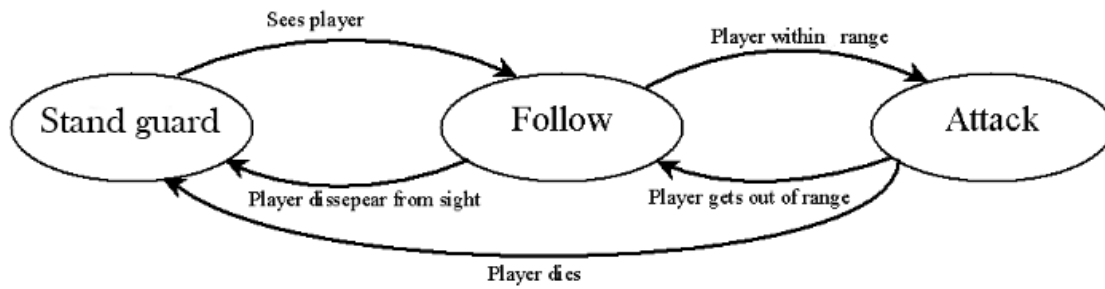


Figure 3-13: A simple finite state machine for boss

```

public enum BossState { avoid, attack, attack_faster };
public static BossState currentState = BossState.attack;

```

Figure 3-14: FSM states for boss

```

public override void Update(GameTime gameTime)
{
    Console.WriteLine(currentState);
    switch (currentState)
    {
        case BossState.avoid:
            UpdateAvoid(gameTime);
            break;
        case BossState.attack:
            UpdateAttack(gameTime);
            break;
        case BossState.attack_faster:
            UpdateAttackFaster(gameTime);
            break;
    }
}

```

Figure 3-15: FSM Implementation (boss)

```

public enum GameState { MainMenu, Tutorial, Gameplay, GameOver, Win }

```

Figure 3-16: FSM states for game

```

protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    base.Update(gameTime);

    switch (_state)
    {
        case GameState.MainMenu:
            UpdateMainMenu(gameTime);
            break;
        case GameState.Tutorial:
            UpdateTutorial(gameTime);
            break;
        case GameState.Gameplay:
            UpdateGameplay(gameTime);
            break;
        case GameState.GameOver:
            UpdateGameOver(gameTime);
            break;
        case GameState.Win:
            UpdateWin(gameTime);
            break;
    }
}

```

Figure 3-17: FSM Implementation on Update()

```

protected override void Draw(GameTime gameTime)
{
    base.Draw(gameTime);
    switch (_state)
    {
        case GameState.MainMenu:
            DrawMainMenu(gameTime);
            break;
        case GameState.Tutorial:
            DrawTutorial(gameTime);
            break;
        case GameState.Gameplay:
            DrawGameplay(gameTime);
            break;
        case GameState.GameOver:
            DrawGameOver(gameTime);
            break;
        case GameState.Win:
            DrawWin(gameTime);
            break;
    }
}

```

Figure 3-18: FSM Implementation on Draw()

4.0 Game Development Issues

4.1 Cool Features

1. Powerup

This powerup can change the bullet pattern of the player.

2. Sound effect

We added sound effects when player or enemy shoots.

3. Pattern movement

The enemy2 moves in a pattern, which is based on our predefined list of control points.

4. Challenging boss

Our boss has the ability to dodge player's bullets and the ability to attack faster during low health.

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

We added a powerup after the completion of level 1 (defeating the boss at the end of level 1). The powerup changes the bullet pattern of the player's avatar.

2. Tutorial (pop-up)

A simple tutorial on the game input is provided at the beginning of the game.

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.

4. Player's health

We added 5 more health to the player's health because we felt that having 5 health was too challenging. The current health for player is 10 instead of the original 5.

5. Player's fire rate

We also reduced the player's fire rate from 150.0f to 300.0f so that the player's avatar will not be too strong since we already buffed its health.

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)

1. Collision between enemies

Currently, we did not implement pathfinding or collision check between enemies. If we had more time, we can make sure that the different types of enemies will not collide with each other since it does not look as nice.

5.0 User Manual

5.1 How to play?

5.1.1 Main Menu

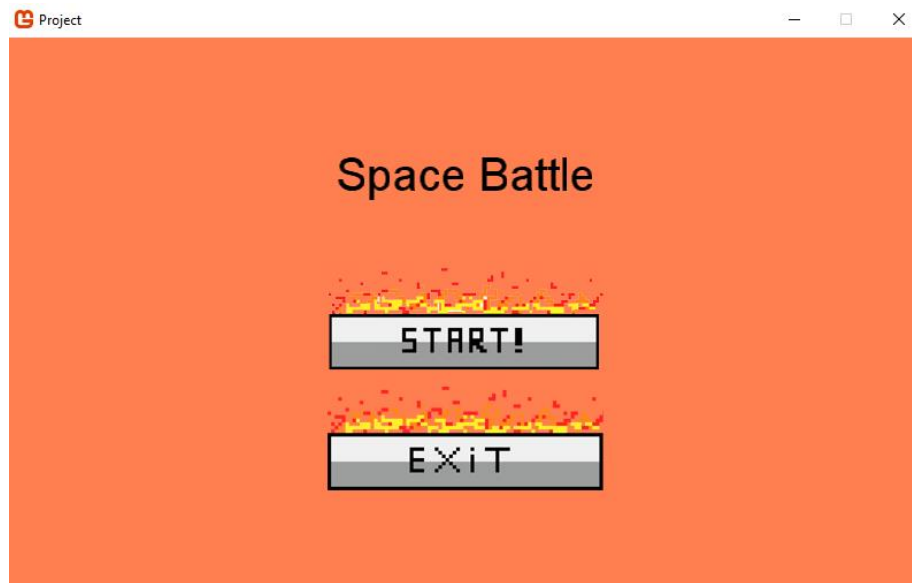


Figure 5-1: Main Menu Screen

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.

5.1.3 Tutorial

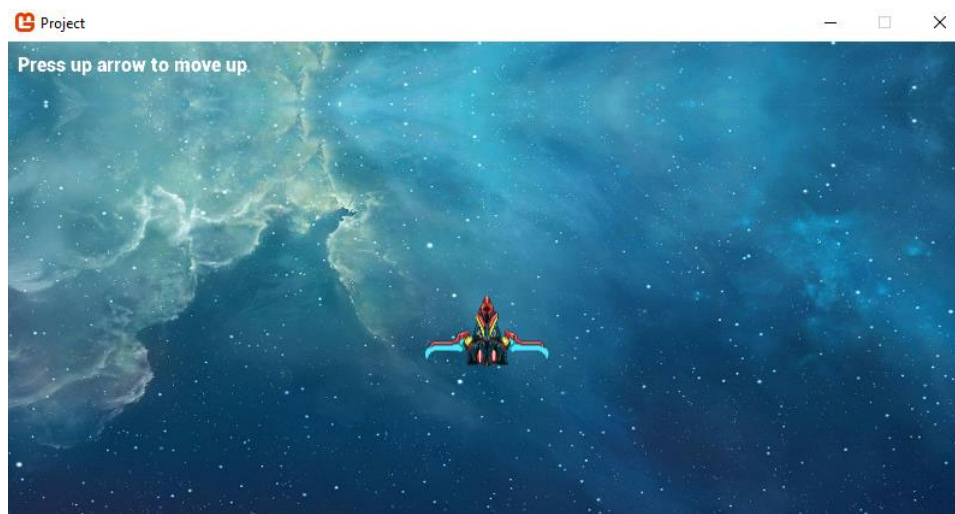


Figure 5-2: Tutorial Screen

If the player clicks on START! button on the Main Menu Screen, the game will start first with the tutorial. The instruction is displayed on the top left corner of the screen. After the player presses the up-arrow key, then the next instruction will be displayed. The tutorial makes sure that the player is aware of all the game inputs; left arrow, bottom arrow, up arrow, right arrow, space bar, "z" and tab. After the completion of the tutorial, the player needs to press "Enter" key before the game will begin.

5.1.3 Gameplay HUD

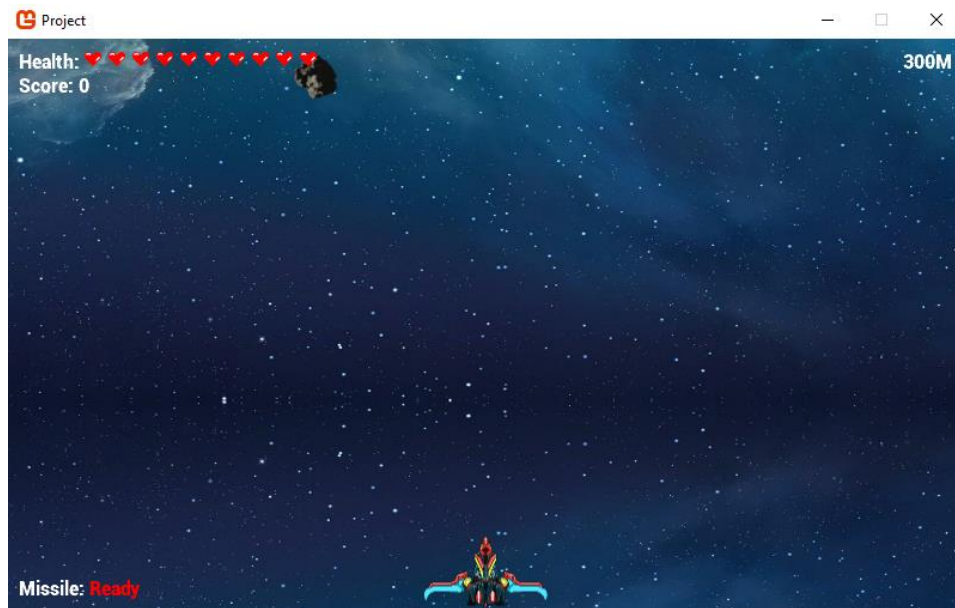


Figure 5-3: Gamplay Screen

After the tutorial is completed, the game will start. On the top left, the player's health (10 hearts) and the score are displayed. On the top right, the distance can be seen too. Each boss will appear at the distance of 2000M. On the bottom left, the missile cooldown is displayed. When the missile is ready to be fired, the message "Ready" will be displayed in red. As you can see from the below figure, the missile is on cooldown of 5s.

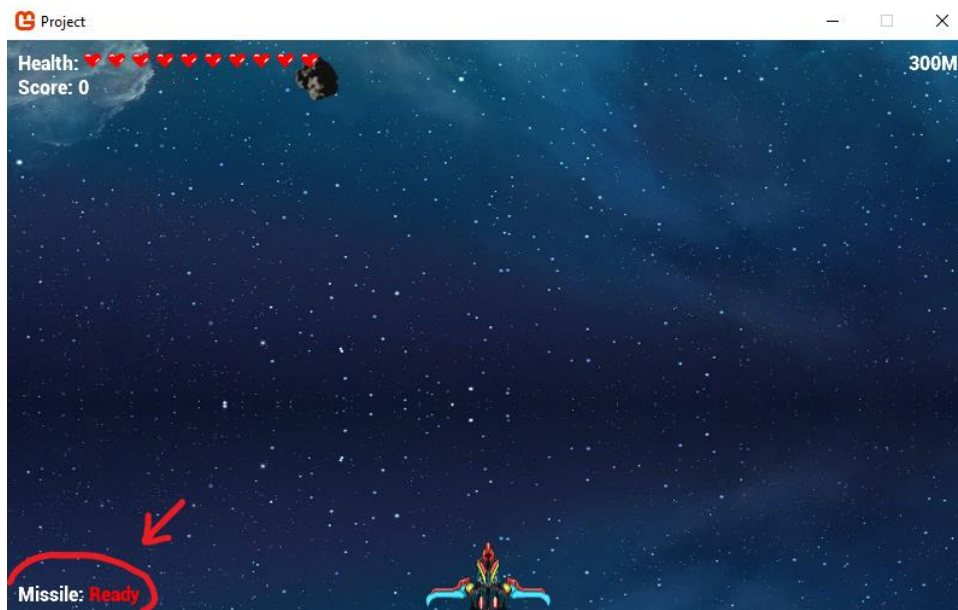


Figure 5-4: Missile Cooldown

5.1.3 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 |

Table 5-1: Game Input Table

As shown from the table above, the player moves around the map by using the arrow keys. Pressing space bar will fire a bullet. Pressing z will initiate the missile for launching and releasing z will fire the missile at that enemy. For missiles, players may select different enemy by pressing the tab button. Pressing the enter button will pause the game and display the pause screen

5.1.4 Pause Game

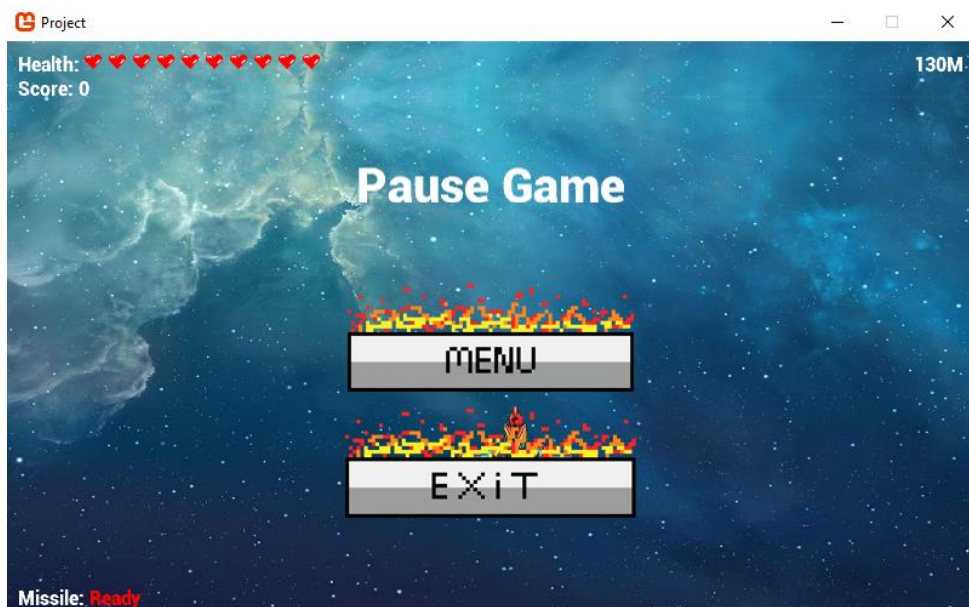


Figure 5-5: Pause Screen

If the player presses ENTER key at any time during the game, the game will be paused, and the screen above will be displayed. Clicking on the MENU button will lead the player to the Main Menu. Clicking on the EXIT button will close the game.

5.1.5 Win



When the player finishes level 1 and level 2, this screen will be displayed, showing that the player has won.

5.1.6 Game Over



Figure 5-7: Game Over Screen

When the player loses all their health, the game will end, and the above screen will be displayed. Clicking on the RESTART button restarts the game from level 1. Clicking on the EXIT button will close the game.

6.0 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/>