Technical Report

**Game Programming Project – Technical Report**

|  |  |
| --- | --- |
| Name | Low Tong Ee Thom |
| ID | 2202423F |
| Class | P03 |

# Q1: Code Refactoring

**Refactoring 1**

|  |
| --- |
| What is the rationale for this factoring?  *Answer why did you choose to refactor this? Is it to make your code clean, or to improve performance, or make it more readable?*  The first refactoring I did was to separate the states in into different scripts      There are 5 different types in one script alone which can be quite annoying  This can be quite a hassle if programmer are adding more behaviour to the classes here. They might have a hard time finding the scripts needed to change which can slow down their programming speed. So to keep it more organize, I plan to separate the classes into different scripts. |
| Before refactoring (attach codes) |
| After refactoring (attach codes) |

**Refactoring 2**

|  |
| --- |
| What is the rationale for this factoring?  *Answer why did you choose to refactor this? Is it to make your code clean, or to improve performance, or make it more readable?*  Some of the code can be hard to understand at first glance. Developers are require to read through all the code in order to understand what the code is doing. We can reduce the time taken to understand code by using function to encapsulate what the code is doing so that developers know what it is about without reading through all the code. |
| Before refactoring (attach codes) |
| After refactoring (attach codes)    you can see how the update function have lesser and more readable code so that it is easier to understand what the code is doing. Programmers would not have to re-read the code every time if they want to make changes to it. They can go to the specific code they want to change and make the changes there. |

Refactoring 3:

|  |
| --- |
| What is the rationale for this factoring?  Next I want to change refactoring to the PlayerMovement Script as I believe it can be more readable. First is to shift the **HandleInput function** into another function to handle the values of the global variable.  I also decided to add a namespace to the script so as to prevent any duplication and make sure that the other developers know which playermovement is used. |
| Before refactoring (attach codes)  public void HandleInputs()  {  // We shall handle our inputs here.  #if UNITY\_STANDALONE  hInput = Input.GetAxis("Horizontal");  vInput = Input.GetAxis("Vertical");  #endif  #if UNITY\_ANDROID  hInput = 2.0f \* mJoystick.Horizontal;  vInput = 2.0f \* mJoystick.Vertical;  #endif  speed = mWalkSpeed;  if (Input.GetKey(KeyCode.LeftShift))  {  speed = mWalkSpeed \* 2.0f;  }  if (Input.GetKeyDown(KeyCode.Space))  {  jump = true;  }  if (Input.GetKeyUp(KeyCode.Space))  {  jump = false;  }  if (Input.GetKeyDown(KeyCode.Tab))  {  crouch = !crouch;  Crouch();  }//this values can be shift to another function so that developer know  //what this is about.  }  public class PlayerMovement : MonoBehaviour  {  //code for player movement  } |
| After refactoring (attach codes)  public void HandleInputs()  {  // We shall handle our inputs here.  #if UNITY\_STANDALONE  hInput = Input.GetAxis("Horizontal");  vInput = Input.GetAxis("Vertical");  #endif  #if UNITY\_ANDROID  hInput = 2.0f \* mJoystick.Horizontal;  vInput = 2.0f \* mJoystick.Vertical;  #endif  HandleValues(); //have a function that explain what to do.  }  private void HandleValues()  {  speed = mWalkSpeed;  if (Input.GetKey(KeyCode.LeftShift))  {  speed = mWalkSpeed \* 2.0f;  }  if (Input.GetKeyDown(KeyCode.Space))  {  jump = true;  }  if (Input.GetKeyUp(KeyCode.Space))  {  jump = false;  }  if (Input.GetKeyDown(KeyCode.Tab))  {  crouch = !crouch;  Crouch();  }  }  namespace PGGE.Player  { //add the name space here to make the code clearer what this script is for  public class PlayerMovement : MonoBehaviour  {  }  } |

Refactoring 4:

|  |
| --- |
| What is the rationale for this factoring?  I also decided to add namespace to the related code to player so that it is easier to keep track the scripts. |
| Before refactoring (attach codes)    One of the many script that don’t that have not use namespace to differentiate. |
| After refactoring (attach codes)    One of the states that starts using PGGE.Player namespace. |

Refactoring 5:

|  |
| --- |
| What is the rationale for this factoring?  I also notice that there is another finite state machine pattern implemented for the multiplayer script. I decided to separate the files and use name space to make the script organize for further use. |
| Before refactoring (attach codes)    The script are all together under a single script which can be misleading to other developers who are getting started with understanding the code. |
| After refactoring (attach codes)    Keep the scripts separated into different folder so that developers are able to find the script related to player movements. |

Refactoring 6:

|  |
| --- |
| What is the rationale for this factoring?  The variable names in the **Player\_multiplayer script** can be quite hard to read due to the numerous variable names that it contain. To organize this, I can use region to keep the variable relevant so that other developer have an easier time to read it. |
| Before refactoring (attach codes) |
| After refactoring (attach codes) |
| Now developers can close the region so that they don’t have to understand all the unknown variables when reading through the code. |

Refactoring 7:

|  |
| --- |
| What is the rationale for this factoring?  I notice that the update function of the player script could be shorten down so that it contains the necessary information require to understand the code. |
| Before refactoring (attach codes) |
| After refactoring (attach codes) |
|  |

Refactoring 8:

|  |
| --- |
| What is the rationale for this factoring?  The variables for the Player movement Script is abit cluttered. So I decided to refactor it by adding region to tidy it up. |
| Before refactoring (attach codes) |
| After refactoring (attach codes) |
|  |

Refactoring 9:

|  |
| --- |
| What is the rationale for this factoring?  I decided to also make the third person camera much more neater for better code readability. |
| Before refactoring (attach codes) |
| After refactoring (attach codes) |
|  |

Refactoring 10:

|  |
| --- |
| What is the rationale for this factoring? |
| Before refactoring (attach codes) |
| After refactoring (attach codes) |
|  |

# Q2: Describe which Option did you choose to implement and how you implemented it. Provide a link to the video that shows your implementation.

# Q3: Provide a video of the two features in action.

# Q4: Performance Optimization

|  |
| --- |
| Identifying the problem   1. Calculating the **boids speed and direction**     I have notice that the coroutine for calculating the flocking is running all on the main thread and is taking a lot of time to run it. Because of that calculation, it will take a huge amont of time (O(n^2) time complexity) to calculate all the boids as the number of boids increase. (Frame will decrease to 20 - 30 FPS with 1000 boids)   1. The boids     The next major issue would be the moving of boids. As the number of boids increase, the time it takes to **individually update** each boids increase linearly. In the picture, you can see how the number of boids can eventually increase the amount taken for each frame. This will lead to a frame per second drop as a result. |
| **Describe step by step on how you implemented the optimization (Attach code)**  After experimenting with different solutions, the solution I current have uses the **Unity Job system**.   1. Fixing the boids movement   Current the script that is moving the boids (Autonomous script) is running in the mainthread. A faster way is by moving all the boids in parallel so that it does not clog up the main thread. This is by transferring all the code from the Autonomous script to the job system using the **IJobParallelForTransform**. Which allows the job system to move the transform in the worker thread itself.  **Code for the job system:**  [BurstCompile]  public struct MovingMovementObject : IJobParallelForTransform  {  [ReadOnly] public NativeArray<MovementObject> boidsData;  public float deltaTime;  public DataRule rulesData;  public Bounds boxBound;  //problem: why does the boids disappear after a set frame?  public void Execute(int index, TransformAccess transform)  {//the index are the same after all...  MovementObject currentBoid = boidsData[index];    transform.position = currentBoid.position; //ensure it is the current position it was last time  RotateGameObjectBasedOnTargetDirection(currentBoid, transform);  MoveObject(currentBoid, transform);  //currentBoid.position = transform.position;  //boidsData[index] = currentBoid; //update the value  }  private void MoveObject(MovementObject curBoid, TransformAccess transform)  {  curBoid.speed = curBoid.speed +  ((curBoid.targetSpeed - curBoid.speed) / 10.0f) \* deltaTime;  if (curBoid.speed > rulesData.maxSpeed) //cap the next speed  curBoid.speed = rulesData.maxSpeed;  float3 vectorToMove = Quaternion.Euler(0, 0, transform.rotation.eulerAngles.z) \* new float3(1, 0, 0);  float3 currentPosition = transform.position;  currentPosition += (vectorToMove \* curBoid.speed \* deltaTime);  transform.position = currentPosition;  }  private void RotateGameObjectBasedOnTargetDirection(MovementObject curBoid, TransformAccess transform)  {  float3 targetDirection = NormalizeFloat3(curBoid.targetDirection);  //get the normalize value of the target direction  float3 rotatedVectorToTarget =  Quaternion.Euler(0, 0, 90) \*  targetDirection;  //not too sure why they rotate the target direction by 90 degree for this...  Quaternion targetRotation = Quaternion.LookRotation(  forward: Vector3.forward, //want to rotate the object through the z axis  upwards: rotatedVectorToTarget);  //then create a rotation based of the vector.  //from: vector3.up to: rotatedVectorToTarget  transform.rotation = Quaternion.RotateTowards(  transform.rotation,  targetRotation,  rulesData.maxRotationSpeed \* deltaTime); //give out the next rotation  }  private float3 NormalizeFloat3(float3 vector)  {  // Convert to Vector3, normalize, and convert back to float3  return ((Vector3)vector).normalized;  }  }  **Code for the struct** MovementObject  public struct MovementObject  {  public uint id; //unique identifier for each boids  public float3 targetDirection; //target direction to know where the boids should look at next frame  public float speed;  public float targetSpeed; //targe speed to know how much it needs to speed up or slow down  public float3 position; //position for the jobs to know every location of the boids  public MovementObject(uint id,  float3 targetDirection,  float speed,  float3 position  ) //initializer  {  this.id = id;  this.targetDirection = targetDirection;  this.speed = speed;  this.targetSpeed = UnityEngine.Random.Range(4f, speed);  this.position = position;  }  };  Now with this code, our boids are now able to in the job system.   1. Fixing the calculations for the boids |
| Before optimization (Attach screenshot of Profiler) |
| After optimization (Attach screenshot of Profiler) |

# Reflect this learning experience

*In this section, reflect on your learning experience associated with this assignment.*

*What have you learnt? How can you use this learning experience? Did you find it challenging to implement?*