Game 5

Vector Movements

Vector Movements ~ Introduction

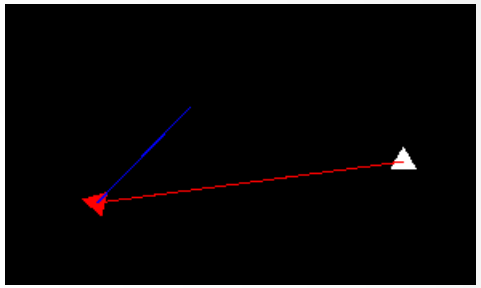
- Vector Movements are an AI technique used to apply realistic physics in games.
- Game objects travelling in a direction should not turn instantaneously. Instead, they should gradually slow down and then turn around.
- Similarly, they should also face the direction they are moving to steadily, rather than suddenly.

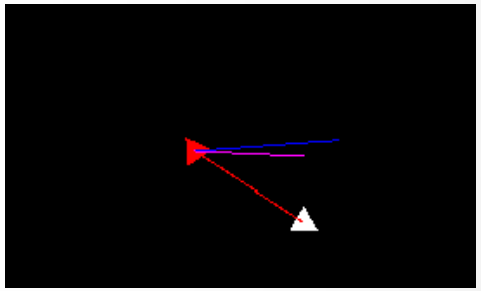
Vector Movements ~ AI Explanation (1)

- There are various different types of Vector Movements. We will be implementing the following:
  - Arrive
  - Avoid
  - Evade
  - Flee
  - Follow the Leader
  - Pursuit
  - Seek
  - Wander

Vector Movements ~ AI Explanation (2)

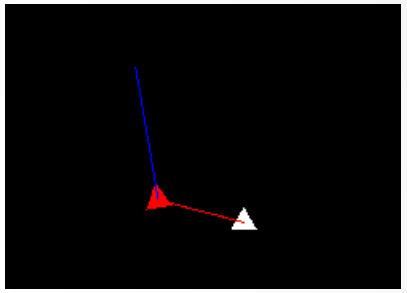
- Seek
  - This type of movement steers moves towards the target's current position.
- Pursuit
  - This movement builds onto seek by moving towards the target's predicted position rather than its current position, this is calculated using the distance between the target and itself as well as the direction the target is facing.
- Note:
  - Blue line represents the direction the game object is currently facing.
  - Red line represents the line to the current position of the target.
  - Magenta line represents the line to the predicted position of the target.

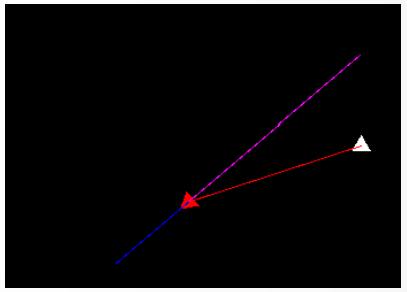




Vector Movements ~ AI Explanation (3)

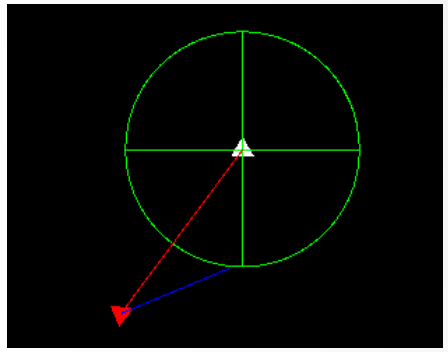
- Flee
  - This type of movement steers and moves away from the target's current position.
- Evade
  - This movement builds onto evade by moving away from the target's predicted position rather than its current position, this is calculated using the distance between the target and itself as well as the direction the target is facing.

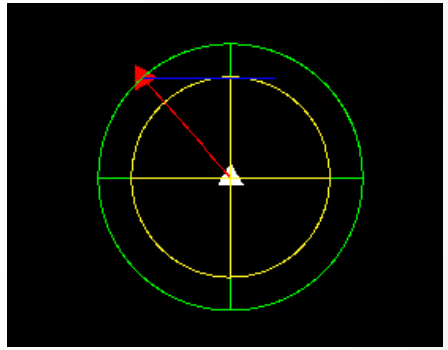




Vector Movements ~ AI Explanation (4)

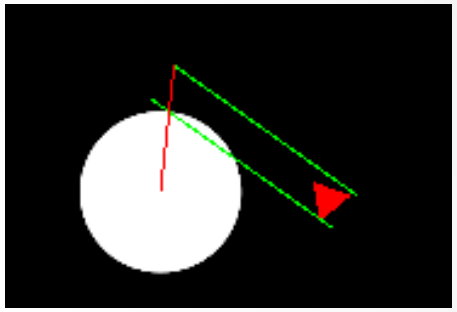
- Arrival
  - This type of movement, similar to seek, steers towards the target and moves towards its current position. However, when it reaches a certain distance away from the target's position, it starts slowing down.
- Follow the Leader
  - This type of movement combines arrival and evasion. When the game object is a set radius away from the target, its performs arrival until it reaches the set radius. If the game object is within a set distance of the target, it evades from it. Thus, the game object tries to stay between the arrival and evasion radius.
- Note:
  - Green circle represents the radius at which the game object starts slowing down. (Arrival)
  - Yellow circle represents the radius at which the game object starts evading from the target's predicted position. (Evade)

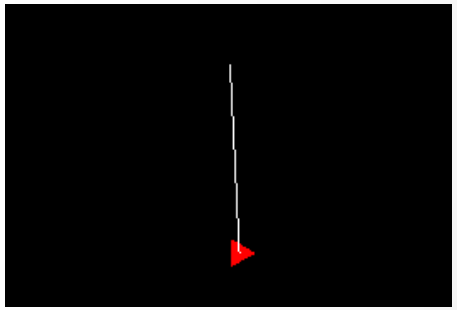




Vector Movements ~ AI Explanation (5)

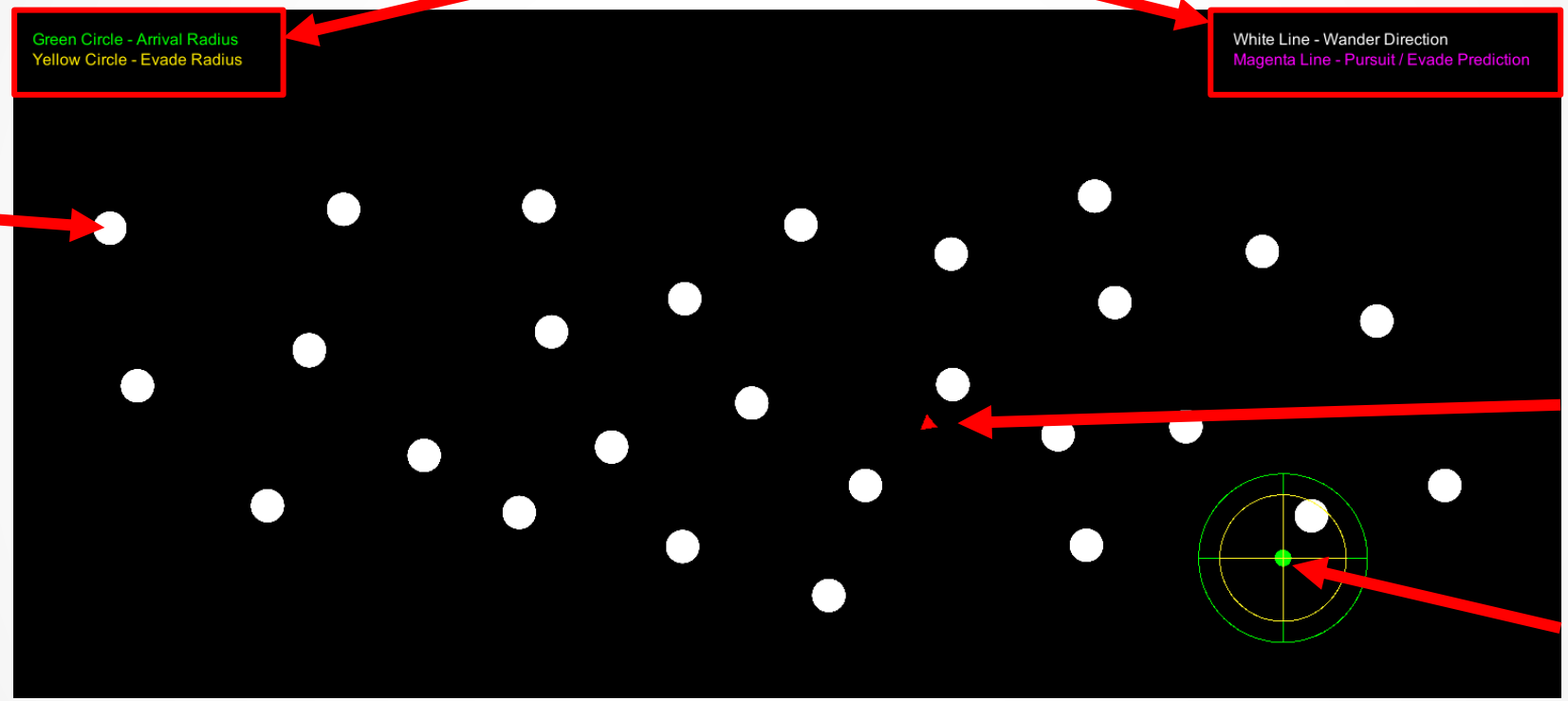
- Avoidance
  - In this type of movement, the game object is given two rays at its sides. If the ray collides with an obstacle, the current object steers away from the direction of the ray it hit. For instance, if its left ray is colliding with a game object, it will steer towards the right.
- Wander
  - This movement decides a random position to move towards for a set amount of time. When the timer reaches zero, another random position is selected.
- Note:
  - Green Rays are the avoidance rays, if they collide with an obstacle, the target will steer away from it.
  - Red line represents the line between the ray and the object it is colliding with.
  - White line represents the direction it is wandering to.





Vector Movements ~ Mini-Game Implementation (1)

Playable Area:



Vector Movements ~ Mini-Game Implementation (2)

- Implementation of the Mini Game was inspired from [1-4].
- The scripts found in the Scripts sub-directory include the:
  - Target – This script handles cursor movement and its visibility.
  - ManageScripts – This script manages the legend and debug lines shown in the game. It also calls transforms the position of the game object to the result of the calculateMove() function call.

Vector Movements ~ Mini-Game Implementation (3)

- The scripts found in the Behaviours sub-directory include the:
  - VectorBehavior – This script serves as the abstract class for the implementation of any vector movement script.
  - CompositeBehavior – This script combines calls all the behavior objects's calculateMove() function attached to it and calculated their sum depending on the weighting assigned to each of them. The calculateMove() function is present in each behaviour object and it is responsible for returning a Vector3 position to move towards.
  - The remaining scripts (ArriveObject, AvoidanceObject, EvadeObject, FleeObject, FollowTheLeaderObject, PursuitObject, SeekObject, WanderObject) perform their respective functionality as indicated by their name.

Vector Movements ~ Mini-Game Implementation (4)

- Each of the remaining behaviour scripts mentioned has the following functions:
  - CalculateMove() – Responsible for calling the other functions and returning the Vector3 to move towards.
  - SteeringFunction() – Calculates and updates the acceleration variable.
  - ApplySteering() – Calculates the Vector3 to move towards using the current velocity and acceleration.

Vector Movements ~ Exercise (1)

- Replace the dummy program with the functionality mentioned earlier. Each type of vector movement is based on a DummyBehaviour script.
- A guide is provided in the next few slides.

```
public override Vector3 CalculateMove(Transform current, GameObject target) {
    //Set location to the game object's position
    location = current.position;
    //Set startPosition to the game object's position
    startPosition = current.position;

    //DUMMY BEHAVIOUR

    Vector3 tempLocation = current.position + (current.up * maxSpeed * Time.deltaTime);

    /*

    SOLUTION GUIDE

    For each behaviour type, you need to first calculate the acceleration.
    After this, you need to calculate the velocity.

    For Seek, calculate the change in velocity to use when calculating the acceleration by subtracting the target's position with the current game object's position
    For Flee, calculate the change in velocity to use when calculating the acceleration by subtracting the current game object's position with the target's position
    For Pursuit, calculate the change in velocity to use when calculating the acceleration by subtracting the target's position with the current game object's position
    plus some prediction vector, based on the target object's distance away from the current game object
    For Evade, calculate the change in velocity to use when calculating the acceleration by subtracting the current game object's position with the target's position
    plus some prediction vector, based on the target object's distance away from the current game object
    For Wander, set its target to a random direction which changes every few seconds
    For Avoidance, use raycast to detect if an obstacle has been detected, note that obstacles are in layer 8, if an obstacle has been detected,
    change the direction of travel until it is no longer being detected
    For Follow the Leader, combine both evade and seek, enabling and disabling the scripts depending on the distance between the target and the current game object
    For Arrival, if the target is within the arrival radius, slow the velocity down

    When the script is done, create a game object by clicking right click, Create -> Behaviours -> BehaviourName
    Then add it to the composite game object, and add its respective weighting

    */

    return tempLocation;
}
```

Vector Movements ~ Exercise (2)

- **Now it's your turn to Code ! – Let's implement some Vector Movements ☺**
- Seek Behaviour
  - In the CalculateMove() function, call the SteeringFunction and pass current and target. Also, return the Vector3 returned by calling the ApplySteering() function.
  - In the SteeringFunction() function, calculate the difference between the two positions and normalize it. Multiply the resulting Vector3 by the maxSpeed variable. Calculate the difference between the calculated Vector3 and velocity, clamping it by maxForce. Add the result to acceleration.
  - In the ApplySteering() function, set velocity to itself added to acceleration, clamped by the maxSpeed. Add velocity multiplied by Time.deltaTime to velocity. Reset acceleration back to the zero Vector and return location.

Vector Movements ~ Exercise (3)

- Flee Behaviour
  - Copy the Seek Behaviour's script and change the subtraction of the positions in the SteeringFunction() function to be the current position minus the target's position.
- Arrive Behaviour
  - Copy the Seek Behaviour's script and in the Steering Function replace the maxSpeed multiplication with the following:
    - Calculate the distance between the target and the location variablne using Vector3.Distance.
    - If the calculated distance is smaller than the maxRadius variable, multiply changeVelocity by distance, otherwise multiply it by maxSpeed.

Vector Movements ~ Exercise (4)

- Pursuit Behaviour
  - Copy the Seek Behaviour's script and add the following:
    - In the SteeringFunction() function, add the Vector3 prediction to the calculated Vector3 before normalising it.
    - Create a new function PredictionMovement() which takes current and target as parameters. In this function, calculate the distance between current and target using Vector3.Distance. Set prediction to the target's up vector multiplied by the distance calculated divided by two.
    - In the ApplySteering(), call the PredictionMovement() function after resetting the acceleration variable, pass current and target.

Vector Movements ~ Exercise (5)

- Evade Behaviour
  - Copy the Pursuit Behaviour's script and change the subtraction of positions in the SteeringFunction() function to be the current position minus the sum of the target's position and the prediction variable.

Vector Movements ~ Exercise (6)

- Follow the Leader Behaviour
  - From the base dummy behaviour, add two public VectorBehaviours, and in Unity set them to the evade and seek behaviour. Also add a public float variable called seperationDistance.
  - In the CalculateMove() function, if the distance between the current position and the target's position is greater than or equal to the seperationDistance variable, return the Vector3 returned from calling the CalculateMove() function on the seek behaviour, by passing current and target. Otherwise, if the distance is smaller than the seperationDistance multiplied by some value, for instance 0.75 (used to create a gap where the game object is neither seeking or evading), return the Vector3 returned from calling the CalculateMove() function on the evade behaviour, by passing current and target. Otherwise, simply return the current position.

Vector Movements ~ Exercise (7)

- Wander Behaviour
  - Copy the Seek Behaviour's script and add the following to the CalculateMove() function:
    - Add the variables below to the script.
    - After calculating the startPosition, decrease the tempTimer by Time.deltaTime. Set centre to the current position added with the multiplication of the current up vector and the seeAhead variable.
    - If the tempTimer has reached zero, set direction to a random direction using direction = Random.insideUnitCircle.normalized; and reset tempTimer to timePerDirection.
    - Set the target's position to the centre added with the multiplication of direction and radius.

```
[SerializeField] private int timePerDirection;
[SerializeField] [Range(5, 10)] private int radius;
[SerializeField] [Range(10, 20)] private int seeAhead;
private Vector2 centre, direction;
private float tempTimer;
```

Vector Movements ~ Exercise (8)

- Avoid Behaviour
  - Copy the Seek Behaviour's script and add the following:
    - Add the variables below to the script.
    - Add a function called CreateBox() which takes current and target as parameters. Set bottomRight, bottomLeft, topRight and topLeft as follows:
      - bottomRight = current.position + (current.right * (sizeX / 2f)) + (-current.up * (sizeY / 2f));
      - bottomLeft = current.position + (-current.right * (sizeX / 2f)) + (-current.up * (sizeY / 2f));
      - topRight = current.position + ((current.right * (sizeX / 2f)) + (current.up * seeAhead));
      - topLeft = current.position + (-current.right * (sizeX / 2f)) + (current.up * seeAhead);

```
[SerializeField] private float maxSpeed, maxForce;
[SerializeField] [Range(1, 10)] private int seeAhead;
[SerializeField] private float sizeX = 1f;
[SerializeField] private float sizeY = 1f;
private Vector3 acceleration, velocity, location, startPosition, topLeft, topRight, bottomLeft, bottomRight;
```

Vector Movements ~ Exercise (9)

- Avoid Behaviour
  - Copy the Seek Behaviour's script and add the following:
    - Create a function called SetBox() which takes current and target as parameters. In the function calculate the current game object's z rotation and save it using eulerAngles.z.
    - Set the current rotation to zero using Quaternion.Euler(Vector3.zero) and then rotate it by zRot using the same function.
    - Create a function called CollisionsCheck which takes current and target as parameters. In the function create an array of two elements of type RaycastHit2D. Set its elements as follows:
      - checkHit[0] = Physics2D.Raycast(bottomLeft, topLeft - bottomLeft, seeAhead, 1 << 8);
      - checkHit[1] = Physics2D.Raycast(bottomRight, topRight - bottomRight, seeAhead, 1 << 8);

Vector Movements ~ Exercise (10)

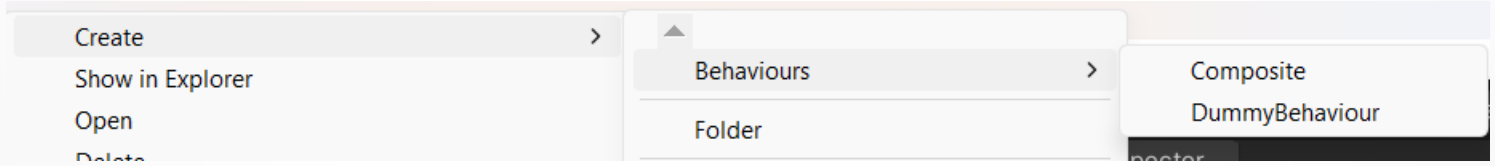
- Avoid Behaviour
  - Copy the Seek Behaviour's script and add the following:
    - In the CollisionsCheck() function, check if the left raycast has collided. If it has calculate the difference between topRight and the collider's position. Multiply this difference b the distance between the current position and the collider's position. Call the SteeringFunction() function and pass the calculated value and current. Else if the right raycast has collided, perform similar functionality but in the opposite direction. Else, call the SteeringFunction() function and pass the current position and the current.
    - In the CalculateMove() function, after calculating the startPosition, call the CreateBox() function and pass current and target. Call the CollisionsCheck() function and pass current and target.
    - In the SteeringFunction() function, replace target with a Vector3 targetPosition.

Vector Movements ~ Exercise (11)

- Finally, replace the following code with the script's appropriate name:

```
//Creating a menu option to simplify Dummy behaviour creation
[CreateAssetMenu(menuName = "Behaviours/DummyBehaviour")]
```

- In the ObjectScripts folder, press right click -> Create -> Behaviours and choose your behaviour.



- Add the created object to the composite object and set its weight appropriately.

Vector Movements ~ Conclusion

- In conclusion vector movements has the benefit of making movement seem less rigid and more smooth. Similarly this makes the game passage more naturally and realistically, providing a better user experience.

Vector Movements ~ References

[1] - Pennywise881, “Steering-behaviors/2d steering behaviors/assets/scripts at master · PENNYWISE881/steering-behaviors,” GitHub. [Online]. Available: https://github.com/Pennywise881/Steering-Behaviors/tree/master/2D%20Steering%20Behaviors/Assets/Scripts [Accessed: 26-Mar-2023].

[2] - “How to make a homing missile in unity with trajectory prediction (source included),” YouTube, 26-Jan-2022. [Online]. Available: https://www.youtube.com/watch?v=Z6qBeuN-H1M&ab_channel=Tarodev [Accessed: 26-Mar-2023].

[3] - Prof. A. Dingli, ICS2211: “LEVEL 2 MOVEMENT” [Online]. Available: https://www.um.edu.mt/vle/pluginfile.php/1103257/mod_resource/content/1/Level2_Movement.pdf [Accessed: 18-Mar-2023]

[4] - “Character follow player Ai - Mini unity tutorial,” YouTube, 05-Feb-2020. [Online]. Available: https://www.youtube.com/watch?v=Mx9M0ieR1M0&ab_channel=TheCodersCat [Accessed: 26-Mar-2023].