# Game 3

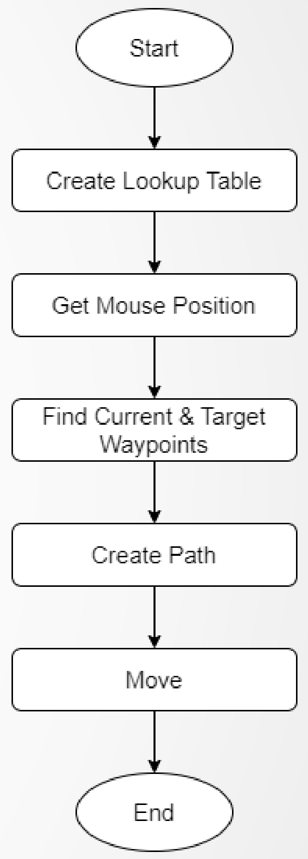
Waypoint Navigation

## Waypoint Navigation ~ Introduction

- Waypoint Navigation is a simple pathfinding technique which uses a series of points, or nodes, to reach a target position.
- It is very useful for small point-and-click games which do not require too much character or AI movement.
- All paths are typically pre-calculated and placed into a lookup table so that the agent can know where to pass from.
- Because of this, it is also computationally expensive on larger maps due to the amount of possible paths it can take.

# Waypoint Navigation ~ AI Explanation (1)

- As soon as the game runs, the lookup table is automatically created using a list of transforms called Points, which are each passed as parameters.
- It loops through the list, and then performs another nested loop through the list. This is done because the lookup table needs to be 2-dimensional so that the agent can later provide a start position and end position.
- After checking whether the two positions are equal to each other, a RaycastHit2D is performed between them. This is done to ensure that there are no obstacles blocking the path between them.



## Waypoint Navigation ~ AI Explanation (2)

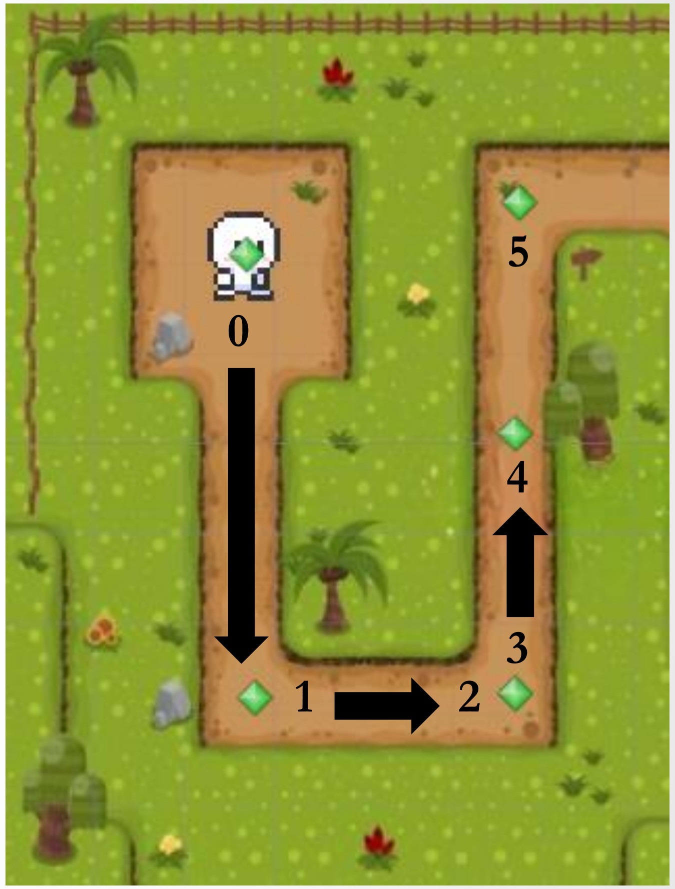
- After the Raycast is fired and a Collider is hit, two scenarios can happen:
  - If the name of the Game Object that the Collider is attached to is the same number as the iteration of the nested loop, then the list is updated to create a path between the two positions.
  - If the name and iteration number are not the same then another nested for-loop is created to iterate either backwards or forwards, depending on the iteration of the start position.
    - If the iteration number of the start position is greater than that of the end position, it iterates forwards, and vice-versa.
    - When it manages to detect a possible path to the end position, the loop ends and the list is updated.

## Waypoint Navigation ~ AI Explanation (3)

- Once all the paths have been created, the game can start taking input from the user through the mouse.
- When a new position is selected, the function FindTargetWaypoint() is executed to retrieve the agent's current waypoint and the target waypoint.
- This function works by iterating through the Points list and finding the closest waypoint to either the player or the target position through the use of Raycasts.

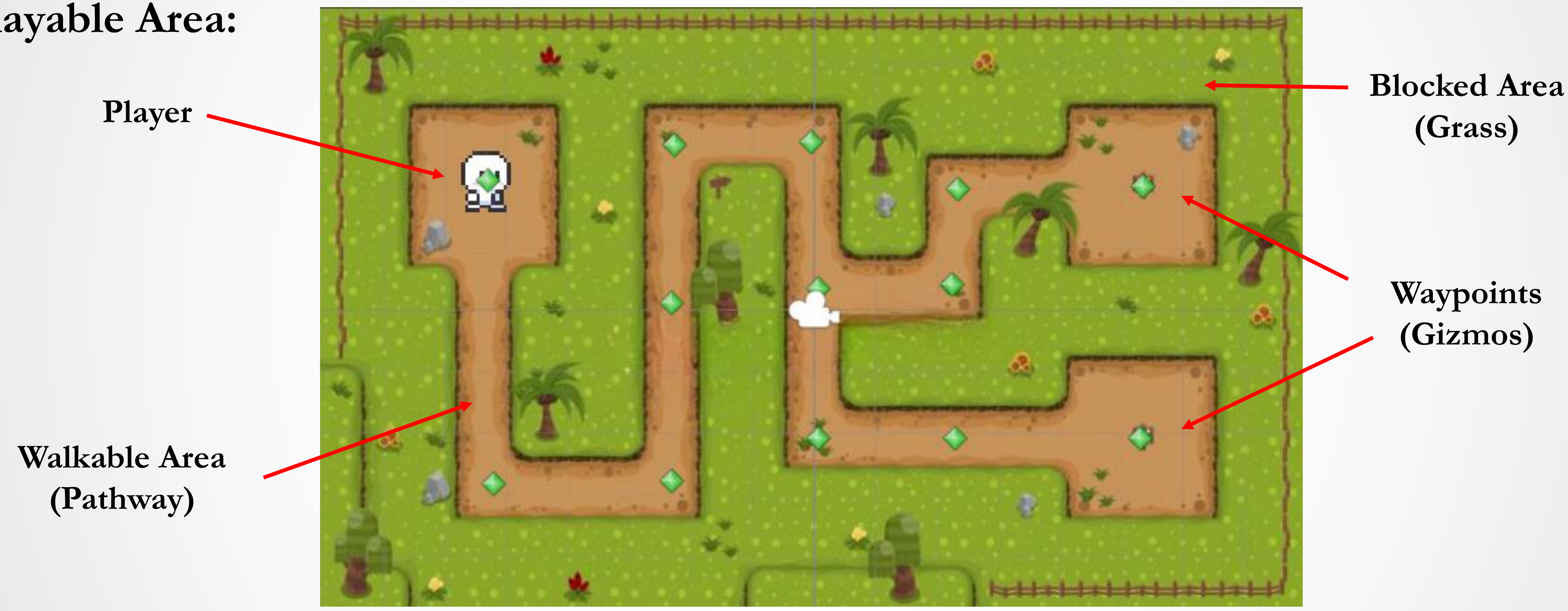
# Waypoint Navigation ~ AI Explanation (4)

- Now that the lookup table is complete, and the current and target waypoints are set, all that's left is to search the table with the waypoints to find the path that needs to be followed. This is performed by the function CreatePath().
- For example, if the starting waypoint is 0, and the target waypoint is 4, then the table is searched with [0, 4], which in this case returns 3 since the player can't reach waypoint 4 directly from 0.
- When the path is created, the player's position is simply updated to follow it by popping each target waypoint from a stack.
- Since the path was pre-defined, it is impossible for the player to get stuck.

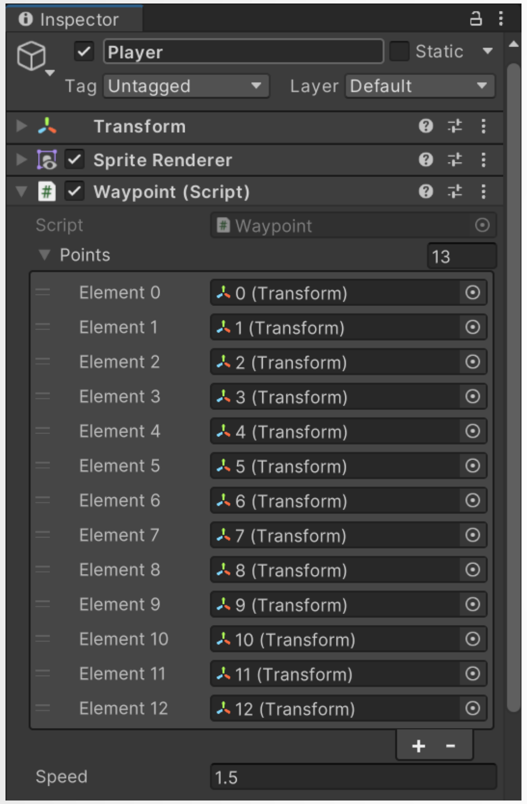


# Waypoint Navigation ~ Mini-Game Implementation (1)

**Playable Area:**



# Waypoint Navigation ~ Mini-Game Implementation (2)



```
Inspector
Player                         Static
Tag Untagged          Layer Default
Transform
Sprite Renderer
Waypoint (Script)
Script          Waypoint
Points                          13
  Element 0     0 (Transform)
  Element 1     1 (Transform)
  Element 2     2 (Transform)
  Element 3     3 (Transform)
  Element 4     4 (Transform)
  Element 5     5 (Transform)
  Element 6     6 (Transform)
  Element 7     7 (Transform)
  Element 8     8 (Transform)
  Element 9     9 (Transform)
  Element 10    10 (Transform)
  Element 11    11 (Transform)
  Element 12    12 (Transform)
Speed           1.5
```

- The waypoints created should be placed as parameters under the Player Game Object into the Points list, where their transforms are taken for the code.

- The speed at which the player moves can also be set from here.

# Waypoint Navigation ~ Mini-Game Implementation (3)

- **How to Set Up Package:**
  - After importing package, navigate to the top right of Unity Editor and do the following:
    - **Layers > Edit Layers.**
    - **Click the Slider button at the top right of the Inspector, in between the question mark and the three dots.**
    - **Select Preset.**
  - Now navigate to the top left of Unity Editor and do the following:
    - **Edit > Project Settings > Physics 2D > General Settings**
    - **Make sure the option 'Queries Start on Colliders' is unticked.**

# Waypoint Navigation ~ Exercise (1)

- **Now it's time to implement this yourself!**

1. Open the Waypoint script and navigate to the CreateLookupTable() function.
2. Create a 2D Array of type Integer. This is where you will store your paths!
3. Open up a loop (i) to iterate through each waypoint from the Points list. Then create a nested loop (j) within it that does the same thing. The first loop will be used for the starting points, while the nested one will be used for the target points.
4. Use a Raycast to find the colliders between the start and end points.
5. Check the name of the Game Object the collider is attached to.

# Waypoint Navigation ~ Exercise (2)

6. If the name of the Game Object the collider is attached to is equal to j:
   - Update the Paths Array at i and j and insert j into it.
7. Else do the following:
   - Compare i to j.
   - If it is smaller, then loop backwards from j until you hit 0, and at each point, use a Raycast to see if you can find the end position like before.
   - If you find it, update the Path Array with the new iteration number you have from this loop and break the loop.
   - If i is greater than j, you must include another nested if-statement to check if i is greater than 10, and if j is between 7 and 9. This is because in that specific range of waypoints, the direction of the player is inverted depending on where they're coming from.
   - If i and j are in between that range, redo the same code from the previous if-statement, else, loop forwards until you reach i instead of looping backwards.

## Waypoint Navigation ~ Conclusion

- Overall, Waypoint Navigation is a very handy pathfinding technique which is easy and quick to implement.
- It is very useful for agents with limited movement, or for maze games such as this one.
- However, it has many limitations and is very computationally expensive.
- Another disadvantage is that it would be tedious to update the waypoints after finishing the code.

## Waypoint Navigation ~ References

[1] – Prof. A. Dingli, ICS2211: "Level3_Pathfinding" [Online]. Available: https://www.um.edu.mt/vle/pluginfile.php/1108327/mod_resource/content/1/Level3_PathFinding.pdf [Accessed: 16-May-2023]

[2] – UnityTutorial World, How to Create Simple Waypoint System for 2D Unity Game? Simple Tutorial (Beginner Tutorial )., 2019 [Online video]. Available: https://www.youtube.com/watch?v=KsePvOltIKM [Accessed: 16-May-2023]

[3] – Goldmetal, "Unity Asset Store: Simple 2D Platformer Assets Pack" 2021 [Online]. Available: https://assetstore.unity.com/packages/2d/characters/simple-2d-platformer-assets-pack-188518 [Accessed: 16-May-2023]