comp476-assignment1

Personal information

COMP476 Assignment #1: Advanced Game Development, Concordia University
Presented by: Thomas Backs
ID: 27554524

Introduction

To explain this program, here a detailed information of each file, basically to swap between different action I used an enumator named Actions which contains: WANDER, ARRIVE, FLEE, PURSUE, TAGGED, RESCUE. There is also another enumerator named Team which contains: RED, BLUE. By default all characters Actions is set to WANDER, for the team, it is setted in the Scene depending to the object it is attached to.

Scripts file explanation

AIRespawn.cs
This file contains the script for respawning the characters to the other end of the map. `public bool x_axis;` , this is used to see if the trigger is on the x-axis or the y-axis.
`public float offset;` , this is used to set an offset when it respawns on the other end to prevent accidental re-trigger. Attached to GameObject in Scene:

- In child of Border game object named: Left, Right, Top, Bottom

BlueAIBehavior.cs and RedAIBehavior.cs
This file contains the script for the AI Behavior of the blue team. The reason it is splitted it is because of easier maintenance and handle particularity if exists of each team. In the end, it is kind of a mistake to have two separate file (one for blue and one for red) since it leads to more overhead and duplication... this variable hold the distance from target: `float distance to target`
There is also 2 boolean as well: `public bool enemy_area = false;` and `public bool has_flag = false;` . There is also `public float speed = 2.0f` that will handle speed of actions. For all actions the characters use this `public GameObject target` and his transform will be used in all movement function.
Attached to GameObject in Scene:

- For Red team: RedPlayer1, RedPlayer2, RedPlayer3, RedPlayer4, RedTester *testing purpose only, it is disable by default*
- For Blue team: BluePlayer1, BluePlayer2, BluePlayer3, BluePlayer4, BlueTester *testing purpose only, it is disable by default*
- Arrive()
    - `public float time2target = 1.0f`
    - `float near_speed = 1.0f;`
    - `float near_radius = 5.0f;
    - `float arrival_radius = 0.5f;`
    - This function handle Arrive action, to be able to activate the function, enumator named current_action need to be set to Actions.WANDER, which is already setted by default. This function is also activated for Actions.RESCUE, the reason for RESCUE usage is to make troubleshooting and testing easier.

- Flee()
    - `public float minimum_distance = 0.5f`
    - This function handle Flee() after the character has tagged another player or have been free'd. This actions only last 1 second before starting to wander again or perform any other actions.
- Pursue()
    - This actions act like Kinametic Seek but with prediction behavior, it will predict the next character position and set it as target in order to prevent the attacking player to reach the flag.
- Wander() - `float max_wander_variance = 0.0f;` - `float wander_offset = 200.0f;` - `float wander_radius = 100.0f;` - `Vector3 current_random_point;` - `float wander_timer_refresh = 0.0f;`
    - This function call another function inside which is called `WanderCirclePoint()`. There is a wander_time_refresh (set at 3.0f) variable which will be used to force wait before refreshing a new random target point, to allow the character to move in a certain direction for a while.
- OnCollisionEnter()
    - This function handle any collision between 2 characters of opposite team that lead to "tag" expected behavior. Once the player has been tagged, an yellowish halo will appear underneath the player and the player will remain in position.
- there is also two public function to handle the Actions enumerator, since I need to check some of these Actions in other script file such as **GameController.cs**, it need to have `public int currentAction()` function to return the integer value of the enumerator. and in my Game Controller, I have a similar enumerator with the same order and I used his integer position value as well to compare it. The other function allow me to set a new actions from other script file such as **GameController.cs** as well named `public void setAction(int action)`, it uses a value integer as index position for the enumator.

FlagBehavior.cs
This script contains all behavior related to flag itslef, such sa flag behavior when it is getting captured and returned to its position. It has a Capsule collider with a trigger to trigger event. It uses `public GameController game_controller` script to be able to use the `flag_captured` boolean in the **GameController.cs** to alert all characters that a flag has been captured by a character. It also uses `public GameObject enemy;` to be abled to "attach" itself to the character who capture it, basically it will follows him to his based unless that enemy is TAGGED, then it will automatically return to its home base. It has `public bool flag_captured = false` as well which is acting differently from the one in. **GameController.cs**, this one is for his `Update()` function, it will allows to see if the enemy player actions has become `Actions.TAGGED` then the flag_capture will be switch to false again, and it will returns to his default home position. There is also `public enum Team {RED, BLUE};` as well as `public Team team;` enumerator, it will allow re-usage of the same script for both team by only selecting the team which this script is attached in Scene.

- OnTriggerEnter()
    - This function triggered itself when an enemy player hit it, it will change the target of the enemy player to his own home base, will change the `flag_captured` variable for BOTH Game Controller and itself to `true` and change the enemy player variable `has_flag =`

`true`. And set its own flag position to enemy position.
Attached to GameObject in Scene:

- BlueFlag, RedFlag,

GameController.cs
This file contains these functions to determine Attacker, Defender, and if the game is over. It is acting like as coordinator between all characters behavior, it will also set and determine the proper actions to do for each character.

- Attacker(): have a random range between 0 and 3, to determine the character that is currently in `Actions.WANDER` and change it to `Actions.ARRIVE` to try to grab enemy flag.
- Defender(): have a random range between 0 and 3, to determine the character that is currently in `Actions.WANDER` and change it to `Actions.PURSUE` to try to stop enemy player in their home area from grabbing the flag.
- GameOver(): this is a mechanism implemented for game over and a restart, it will restart after 5 seconds, since the game is currently in inconsistency phase, it will behave differently after each restart and run. *might add a force restart mechanism as well*
- Rescue(): this function allow us to select a player to rescue another tagged player by going throught a list of currently tagged player and selected the closest player available to change its action from `Action.WANDER` to `Actions.RESCUE`. Once a character has been selected to perform this task, it will set `public bool [color]_rescue = true`. Once the character collides with his tagged teamate (this behavior is explained in **RedAIBehavior.cs and BlueAIBehavior.cs**), the actions will be changed to `Actions.FLEE` toward their base, and `public bool [color]_rescue = false` to allow the Rescue() operations to start again to find another tagged player. Inside this function, it will call another function called WholeTeamTagged(int, int).
- WholeTeamTagged(int red_counter, int blue_counter): it will take the tagged player counter from both team and see if the entire team has been tagged, if so, then the game is over for the team that has been tagged and the team with remaining untagged player wins!
- Once the Defender() action is over it will set the `[color]_defender = false`, and if the player is tagged it will also set both `[color]_attacker = false` and `[color_defender = false` with respect to the tagged condition. And then the Game Controller looks for another player to perform these actions if needed. The same apply to Rescue(). Attached to GameObject in Scene:
- GameController

HomeArea.cs
This file handle all player that enter and exit its own area. It will set either `[color]AIScript.enemy_area = true;` if the enemy player enter its home area or `[color]AIScript.enemy_area = false;` if the player exits the area, but the exit part can be confusing because it will **NOT** set it to false, this kind of behiavor allow defender to attack the player until its base.

- OnTriggerEnter(Collider other): handle any required actions about enemy player entering home area
- OnTriggerExit(Collider other): handle any required actions about enemy player exiting home area
  Attached to GameObject Scene:
- BlueHome, RedHome

WinningCondition.cs
This file handle all Winning Condition and the proper actions when the player arrive to characters home area destination.

- OnTriggerEnter(Collider other): This will be triggered and perform the Game Win for the team that bring the flag to its home area. It will also notify the **GameController.cs** to do these change: `game_controller.game_over = true` and `game_controller.[color]_win = true` and the game is ended now. The restart timer is started.

Issues

There is few issues with this game that needed to be improved in future, first of all, there might pathing issue, since the characters has random Wander movement, it might end up in enemy area and still continue to WANDER instead of ARRIVE and become a target for enemy character. This "issue" will be leaved there since it could be used as a "bait" for other character to pursue him and tag him. Allowing his teamate to capture the flag. Since there is a lot of randomization process happening in the game, some run will cause trouble as well. There might be some run where everyone is stalled and nobody is doing anything (rarely happens, only happens once after multiple runs). There is another issue, the characters are not very smart, meaning they can't properly avoid collision on its path, it will need some raycasting in the future to be able to "dodge" enemy on its path. This has become problematic when the characters is tagged and stay in this area. Also, since it use Random.Range(min, max) to assign an action and role, it might be a while before finding the proper characters to perform this action, this kind of behavior can be keep at it since it allows a bit of balance and "advantage" as well.

The FLEE mechanism can lead to some weird issue, when they collide they facing opposite from each other and run but when it activate wander, it turn in place while moving until that stabilized.