Neverwinter Nights AI - Report

Daniel Koltai, Laura Gozzo, Michelle Lathouwers, Taisiia Kosenok

Description of methods in final strategy

Emerging behavior with altar scores

The developed AI is based on an emerging strategy, which gives each player of the team tailored instructions based on current observed factors of the given player.
Given that static strategies are difficult to define precisely with adequate level of adaptation to the game states due to the large number of possible scenarios to consider.
Instead, our method adapts to the enemy's strategy by identifying their weaknesses and capitalizing on them. The AI monitors the movement of enemy NPCs to identify and attack their most vulnerable altars.
Attacking close and vulnerable altars is generalizable and hard to define in a static team level strategy.
This was done by defining the function GetLocScore, which assigns scores to each altar based on current factors in the game relative to each player.
The tailored target selection is performed by applying GetLocScore to each player.

Number of enemies around each altar

The GetNumEnemiesAroundAltar function calculates the weighted number of enemies near an altar in a detection radius of 6. The function iterates over all the creatures in the game and checks whether it is a team member or an enemy. For every enemy in the detection radius around the altar the count of enemies is incremented. The function returns the weighted number of enemies. The function considers the type of enemies as well. We considered Fighters and Masters more dangerous than others. To reflect this, their count around altar were weighted by 2 for fighters and by 3 for the master.

Location scores

For each player, the GetLocScore function assigns a score to each of the altars. The score depends on two factors, the distance between the character and the given altar as and the weighted number of enemies near the altar.
Weights were adjusted to adequately capture the importance of each factor and their scales.

```
float score = 0.0 - GetDistanceBetween(OBJECT_SELF, oAltar) - 20.0 * numEnemies;
```

In calculating the final score of this function, altars that are closest to the player and have the least amount of enemies were prioritized. There were also two conditions checked before calculating a score. If the altar was already claimed by a team member -999.9 would be returned. This low value makes sure that this altar would always be ignored, because if an altar is already generating points, then another team member should not waste its time there. If an altar is completely empty, then 999.9 would be returned which would always make it the preferred altar. This was chosen, because an empty altar would mean that the character would not have to fight to capture an altar, and it is an easy way to generate points. In testing this made a big change, because before this implementation characters would sometimes run past empty altars.

Selection of target

Then, each NPC is made to go to the altar which makes the most sense for him to capture, selecting the altar with the highest score.
GetGoodTarget iterates over all altar scores returning the altar corresponding to the highest score.
Select altar with highest score

Roam team

The Roam team consists of a Master. His goal is to identify empty enemy altars (the ones that are closest to the enemy's spawn) and if there is any, he tries to capture one of them. If all of them are occupied, he goes to the Doubler. The function updates the target every 6 seconds, so if one of the enemy's altars becomes empty, he will have the best position to move from, because the Doubler is in the center and the distance to the enemy altars from there is equal.

Testing

The final code worked without any errors; movement of players were observed to match the aim of implementations.

Performance

The developed AI exhibited a strong performance against the default AI, by defeating it multiple times with a score 1.5 to 2 times larger the default AI's score.

Unselected attempt

Initial positions for each character

We wanted to set the default positions for our team after the first spawn to ensure the best positioning in the beginning of the game. The idea was to split the team as follows: Fighters and Wizards would go to the left and right altars that are closest to the spawn (one Fighter and one Wizard per altar), when Master and two Clerics would try to capture the Doubler. The problem with this function was that the team members stood still after taking the default positions and did not move, even if other positions were more preferred at the later stages of the game. Moreover, some of them would still go to the default positions after the second spawn.

Direct master with low health to healers

The goal of this function is to navigate Master to Clerics, when the Master's health is below 30%. The Master is directed to the closes Cleric. If both Clerics are dead the Masters act as if it had a health score above 30%. The AI with the implementation of this function performed worse, because the Master always prioritized healing, even when capturing a nearby empty altar to collect points was possible. Moreover, sometimes Clerics were too far away, and the Master would spend too much time on getting close to one of the Clerics and utilizing itself to fight for targets. This function can potentially be very useful, however a more complex approach is necessary, to determine priorities more efficiently.

Testing

The functions worked, but the performance of the AI was worse with these implementations upon testing both modifications individually and altogether.

Performance

The performance of the code with these functions was still better than the default AI, but in comparison to the same code, but without these functions, it performed worse. So, we decided to not include them in the final version of the code.

Other ideas that we could not implement

Game states

Our idea was to implement a framework based on 3 distinct game states. This framework depends on which team is scoring more points at the current heartbeat and whether there is an empty altar. First, we check for unoccupied altars. If there are unoccupied altars, we deploy creatures to take it. Otherwise, we

either attack or defend. When we score more, we want to hold on to our occupied altars by defending them. When scoring less we want to take enemy altars.

Keeping a player on altar

We want to ensure that there is always an NPC staying in the altars our team currently claims.
We want to do this by marking the players to the waypoint of the altar to always stay there. Other players are available for deployment.

Work distribution

Daniel	Implemented GetLoc(), getGoodTarget(), worked on game states and how to keep a player on altar while deploying others
Laura	Implemented GetLoc(), getGoodTarget(), worked on game states and how to keep a player on altar while deploying others
Michelle	GetNumEnemiesAroundAltar() function, Healing function, improve GetLoc() function,
Taisiia	GetNumEnemiesAroundAltar() function (type of enemies implementation), Roam team function, Initial positions function