Project 2
Decision Making

1 Level 1 - Behavior Trees

1. The selector we developed relies heavily on the sequence. The primary distinction lies in its behavior: it returns a success if any of the tasks return success and a failure if all of them fail.

2. [**Bonus level**] In our initial attempts with the behavior tree, we experimented with using either "moveTo" or "pursue" and the provided patrol points. However, a significant issue arose when the orcs did not interrupt their patrol to chase the player. To address this challenge, we introduced a new task called "PatrolAndPursue." Essentially, this task involves a form of movement that intelligently selects its target. It returns success when it approaches the player closely enough for the orc to initiate an attack. If it doesn't get close enough, the task continues running. This approach effectively combines both patrolling and pursuing into a single task that can dynamically update the orc's movement target each time it's called.

3. For the shout we decided to update the PatrolAndPursue task. Inside the code we already had the check for when the Orc sees the player so inside there we added

```
1 this.myMonster.shout.PlayOneShot((AudioClip)Resources.Load("OrcShout2sec"));
2 ChangeOrcsCurrentTarget();
```

   So that when it starts a chase the first line will play a shouting sound and the second one is a function that will notify all Orcs about the shout so that they can follow it as well.

4. As it was explained in the point before, because of the way the patrolling was implemented it was not necessary to modify the behavior tree. There probably is a way to implement it as a separate action but that would mess with the ability to change from patrolling to pursuing (At least with our current solution for it).

5.

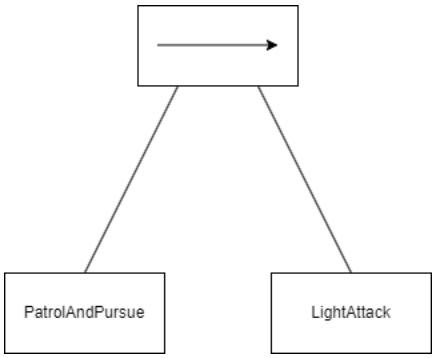

Figura 1: Patrol behaviour tree diagram

2 Level 2 – GOB and GOAP

This section was written just after finishing GOAP and GOB implementation. Therefore it is using old distance calculation and monsters' statistics are unchanged.

2.1 GOB

Best Sir Uthgard's performance

First few trials where unsuccessful, because Sir Uthgard did not focus much on killing monsters. Which was problematic especially when on his way to potion/chest was a monster. It caused Sir Uthgard to get damage and no XP points, therefore he was dying quickly with no visible progress. To modify his behaviour we've raised the importance of the Gain Level Goal, weight setup is presented in table 1. Increasing the weight of gaining level, motivated Sir Uthgard's to kill monsters in order to gain XP. Thanks to this he avoided unnecessary damage, because now he was fighting the mosters and was using DivineSmite on skeletons. Furthermore he started leveling up which increased his HP max level, which was necessary for facing the dragon.

Goal name	Weight
Gain level	3
Survive	2
Be quick	1
Get rich	0

Tabla 1: Weights for GOB algorithm.

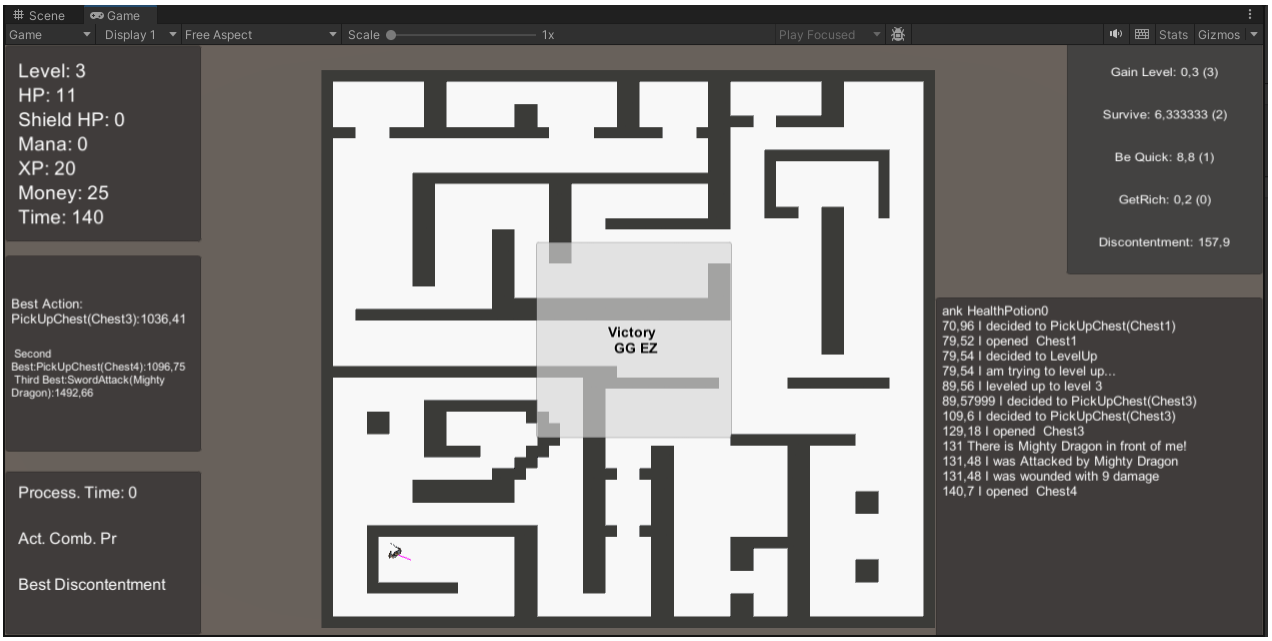

Figura 2: Best Sir Uthgard's performance with GOB algorithm.

2.2 GOAP

Best Sir Uthgard's perfomance

Surprisingly it was harder to win for Sir Uthgard with the GOAP algorithm than with the GOB. The biggest problem was to choose the best Survival Goal weight, choosing too big weight was causing Sir Uthgard to run away from monsters the moment in which they entered the visibility range. He was changing his mind constantly *ex. He decided to attack an orc, but when he was near the orc he would change his decision.* That behaviour was causing him to die quickly or making him run out of time.

Goal name	Weight
Gain level	3
Survive	2
Be quick	2.5
Get rich	0.8

Tabla 2: Weights for GOAP algorithm.

Answers to GOAP questions

1. *Why is Sir Uthgard attacking orcs with very low HP?*
   This issue appears when Sir Uthgard's Survive Goal's weight is too low. However as mentioned above it is not that easy to find proper weight, because increasing it too much causes Sir

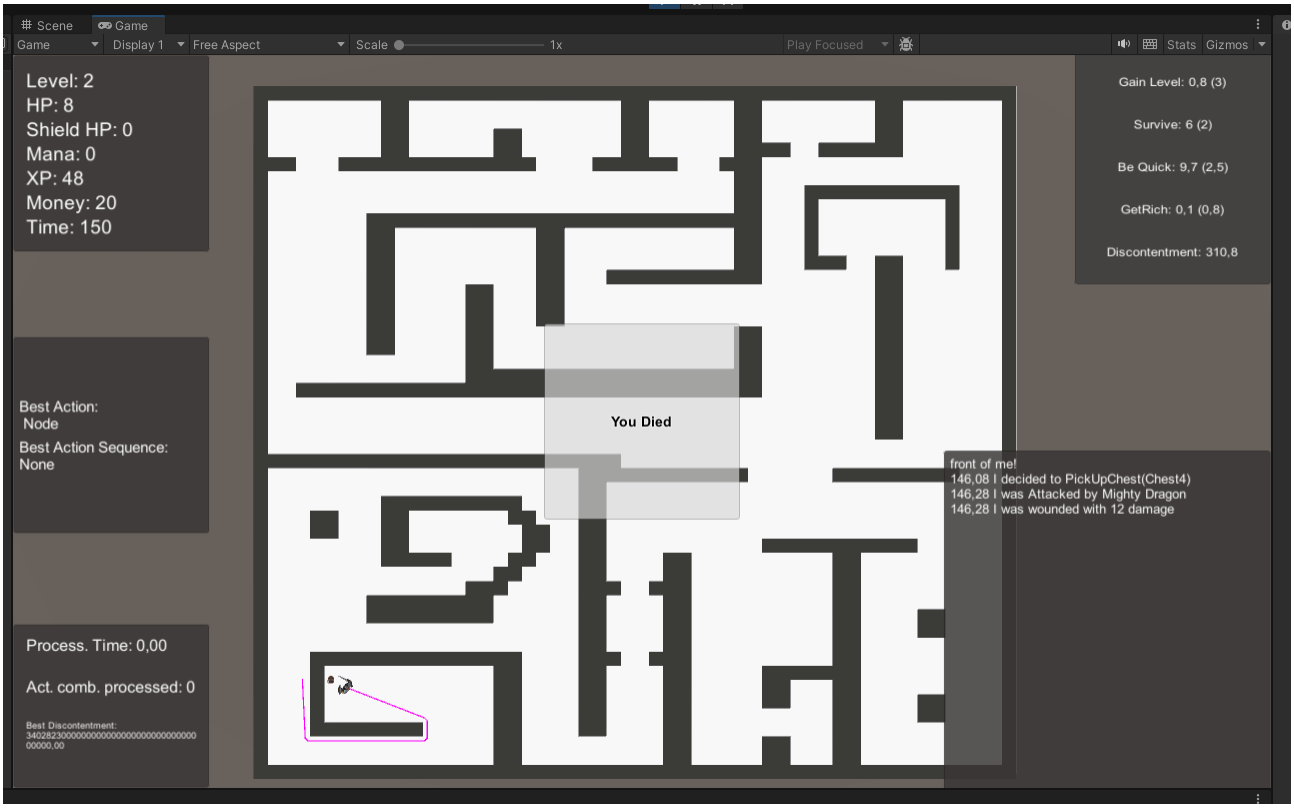

Figura 3: Best Sir Uthgard's performance with GOAP algorithm.

Uthgard to avoid fighting with all the monsters.

2. *How to make Sir Uthgard choose the targets that are close to him?*
   It can be influenced by increasing the weight of Be Quick Goal.

3. *How the algorithm changes when we increase depth of the tree from 3 to 4?*
   Increasing the depth increased the intelligence of Sir Uthgard. As expected the number of processed combinations was much greater. From 4860 for depth 3 it increased to around 80000 for depth 4 (at the beginning of the game), also the processing time vastly increased, from around 150 ms for depth 3 to around 2.5s for depth 4.

3 Level 3 – Sir Uthgard Actions

3.1 General implementation of actions

Even though there are some differences in the implementation of the actions, they follow a similar recipe. We worked on the implementation of these functions:

- CanExecute(): Here we check if the current conditions are right for executing the action.
- CanExecute(WorldModel worldModel): Same as CanExecute() but using the stats of a world model
- Execute(): A simple function to execute the correct action
- GetGoalChange(Goal goal): How the action affects the goals
- ApplyActionEffects(): The changes the actions has on a world model
- GetHValue(WorldModel worldModel): A function that returns a value for choosing better actions on the MCTSBiased playouts. It is generalized for most walkToTarget actions so we can just use the base one.

We also had to add the actions to Sir Uthgard's brain in AutonomusCharacter class. For actions that require a game object we associate them with each one of them, for example for the mana potion:

```
foreach (var potion in GameObject.FindGameObjectsWithTag("ManaPotion"))
{
    this.Actions.Add(new GetManaPotion(this, potion));
}
```

And for actions that only depend on Sir Uthgard we just add them directly, like rest:

```
this.Actions.Add(new Rest(this));
```

For the heuristic we decided to build an "universal" method that would work for most actions (Especially the walkToTargetAndExecute ones). For that we first implemented on the base action the following methods:

```
public static float GetHValueFinal(WorldModel worldModel)
{
    var MAX_HP = Convert.ToInt32(worldModel.GetProperty(Properties.MAXHP));
    var HP = Convert.ToInt32(worldModel.GetProperty(Properties.HP));

    var money = Convert.ToInt32(worldModel.GetProperty(Properties.MONEY));
    var time = Convert.ToInt32(worldModel.GetProperty(Properties.TIME));

    if (HP <= 0 || time >= 150)
    {
        return 1000;
    }
```

```
13      if (HP > 0 && time < 150 && money == 25)
14      {
15          return -1000;
16      }
17      return -HP / MAX_HP - money / 25;
18  }
19
20  public virtual float GetHValue(WorldModel worldModel, float duration)
21  {
22      return Action.GetHValueFinal(worldModel) + duration / 150;
23  }
```

This will make sure the heuristic returns 1000 when we lose, -1000 when we win and if not do a quick calculations similar to the one we made for secret level 2. Time is not initially taken into account because the GetHValue adds it. Then inside WalkToTargetAndExecuteAction we run the heuristic like this (Using Euclidean distance):

```
1  public override float GetHValue(WorldModel worldModel)
2  {
3      var duration = this.GetDuration(worldModel);
4      return base.GetHValue(worldModel, duration);
5  }
```

That way we take into acount the duration of the action. Then in most actions we just use the base GetHValue from this class.

3.2 GetManaPotion

For mana potion, the conditions is just that mana < Max_mana. The goal changes are not existing, and the actions make it refill mana of Sir Uthgard.

3.3 DivineSmite

In DivineSmite we need to have 2 or more mana, the goal changes using the amount of exp we earned (Only more than 0 if the target is a skeleton) . Lastly it makes the character lose 2 mana and win the exp if the attack was successful.

3.4 ShieldOfFaith

Similar to mana potion we check if we have more than 5 mana, the survive goal changes (In the amount of shield we gained) and in the world model we update both the mana + the shieldHP.

3.5 Rest

For rest the precondition is he has damage (Hp ¡ Max HP). Then we change both the survive goal and the be quick goal, given that we change the hp and it takes time. We apply this changes to the world model as well and finally for the heuristic we set the duration to 10 (So it does not do it that often).

3.6 Teleport

For teleport we need the mana and we decided to not change the goals. In the world model we update the position of the player to initial position.

4 Level 4 - MCTS and Biased MCTS

4.1 MCTS Biased Heuristic

Crucial aspect of the MCTS Biased was to define the "GetHValue()" function. The lower the HValue the better action we have. We decided to punish Sir Uthgard severly for dying to make him avoid choosing actions that will cause him to lose the game. Standard value returned by this function put emphasis on keeping good HP and money ratio. Detailed modifications and adjustments for the HValue for spesific actions are described in Level 3.

```
1  public virtual float GetHValue(WorldModel worldModel)
2  {
3      var MAX_HP = Convert.ToInt32(worldModel.GetProperty(Properties.MAXHP));
4      var HP = Convert.ToInt32(worldModel.GetProperty(Properties.HP));
5
6      var money = Convert.ToInt32(worldModel.GetProperty(Properties.MONEY));
7      var time = Convert.ToInt32(worldModel.GetProperty(Properties.TIME));
8
9      if (HP <= 0 || time >= 150)
10     {
11         return 1000;
12     }
13     if (HP > 0 && time < 150 && money == 25)
14     {
15         return -1000;
16     }
17     return - HP / MAX_HP - money / 25;
18 }
```

4.2 Distance calculation - Problems and solution

When running MCTSBiased, even when the enemies were sleeping, we could not win consistently because even though he decided to al pick up chests, he usually did it in the wrong order. It was especially noticeable how after picking chest 3 or 4, he did not go for the other one that was really nearby. Too see what was causing it we decided to take a look at how the duration was calculated, and the most important part of it is the distance. We built this matrix calculating the distance between the chests using the function GetDistanceToTarget given in the project:

	C0	**C1**	**C2**	**C3**	**C4**
C0	68,3	133,7	61	124,4	127,6
C1	63,4	128,8	56,1	119,5	122,7
C2	63,8	129,1	56,4	119,9	123,1
C3	54,2	119,6	46,9	110,3	113,5
C4	69,8	135,1	62,4	125,8	129

Tabla 3: Chest distance using base GetDistanceToTarget()

With these results we can see some issues like the distance between a chest and itself not being 0 or even small in some cases, it not being symmetrical and also depending primarily on the first

coordinate given. To try and solve these we decided to implement a version of Euclidean distance, and we settled with the implementation below (The multiplier is to account for it not having a straight path):

```
1
2 public float GetDistanceToTarget(Vector3 originalPosition, Vector3 targetPosition)
3 {
4     return Vector3.Distance(originalPosition, targetPosition) * 2.5;
5
6 }
```

And with this implementation we got the following distance matrix:

	C0	C1	C2	C3	C4
C0	0	42	41.4	183.8	246.6
C1	42	0	68.3	155.6	218.3
C2	41.4	68.3	0	172.2	233
C3	183.8	155.6	172.2	0	62.9
C4	246.6	218.3	233	62.9	0

Tabla 4: Chest distance using Euclidean Distance

4.3 Testing

Because the MCTS algorithm does not really take into account the patrolling orcs, they tend to kill Sir Uthgard on his way to complete actions. We decided to test Sir Uthgard's behaviour with sleeping enemies and him not changing his action when he sees an enemy. Initial attempts were often not successful because of too low number of iterations. Sir Uthgard was not going for the chests and the Q/N values were really small. Therefore we changed the MaxIterations to 5000 and the Playout numbers to 5 and we got much better results, he manages to win or gets close consistently. It is not 100% mainly due to how the distance is calculated (Euclidean metric) it can severely underestimate the path he will need to take and he thinks he has time to do other action in between.

In table 5 we posted results for different number of playouts for the MCTS and Biased MCTS algorithms. As we can see with high number of playouts, high number of iterations and in simplified world with sleeping enemies and Sir Uthgard not changing decision when he sees a monster. Both MCTS algorithms are usually leading him to win. Especially MCTS Biased is performing well with 100% win ration for small and high number of playouts.

	Avg. HP at the end	Avg. mana at the end	Avg. XP	Avg. time	Avg. money	Avg. level	Win ratio	No. Play.	Max Iter.
MCTS Biased	10.0	2.0	0.6	131.2	25.0	1.0	1.0	1	5000
MCTS Biased	10.0	0.0	0.0	112.0	25.0	1.0	1.0	5	5000
MCTS	4.0	0.0	8.6	124.0	14.0	1.0	0.2	1	5000
MCTS	10.0	0.0	0.0	129.6	24.0	1.0	0.8	5	5000

Tabla 5: Results for MCTS varations with sleeping enemies and Sir Uthgard not changing action when enemy is near.

4.4 Summary

For this specific game the MCTS algorithm is very ineffective. Firstly for it to be succsessful it requires high number of iteration and playouts, which is highly time consuming, it is even more time consuming if Sir Uthgard changes his decision each time he sees an enemy. Also while choosing an action the MCTS algorithm does not consider obstacles like monsters on the way to target, therefore Sir Uthgard is constantly wasting time.

5 Level 5 – Orcs Formation!

Due to time constraints we were not able to implement Level 5.

6 Secret Level 1 – Limited Playout MCTS

Due to time constraints we were not able to implement Secret Level 1.

7 Secret Level 2 - Limited Playout MCTS

For deciding how to do the heuristic first we decided what were the important parameters that we had to analyze and include in the function. We opted to take in to consideration mainly: Current player health Money Time Then assuming a won game gives 1 and a lost game gives 0, we needed to factor in these 3 components in a way that we could get values in this range. Because we want more HP and money, these 2 values will add to the score; and because we want less time these one will subtract. We normalize the 3 of them so they go from $0 \rightarrow 1$ (Dividing them by their max possible value) and we get:

```
return HP / MAX_HP + money / 25 - time / 150
```

But these heuristic goes from -1 (Worst case even though not really reachable) to 2 (Best case although it also it is not reachable), so to fix this we shifted it by 1 so it goes from $0 \rightarrow 3$ and then divided by 3. In the end we get:

```
return (HP / MAX_HP + money / 25 - time / 150 + 1) / 3
```

Additionally we also wanted to take in to account states that he wins or loses completely, so we reused the get score function were we had already taken into account these factors, and put it inside an if after the win/lose check that makes sure it only runs if the state is not a terminal one (So that it does not mess up with the original function). This is the final result:

```

public override float GetScore()
{
    int money = (int)this.GetProperty(Properties.MONEY);
    int HP = (int)this.GetProperty(Properties.HP);
    int MAX_HP = (int)this.GetProperty(Properties.MAXHP);
    float time = (float) System.Convert.ToDouble(this.GetProperty(Properties.TIME));


    // TODO : Should Time and other factors be taken into accoun?

    if (HP <= 0 || time > 150) return 0.0f;
    else if (money == 25 && time <= 150)
    {
        return 1.0f;
    } else if (!this.IsTerminal()) //Non terminal score check for playback
    {
        return (HP / MAX_HP + money / 25 - time / 150 + 1) / 3;
    }
    else return 0.0f;
}
```

	Avg. HP at the end	Avg. mana at the end	Avg. XP	Avg. time	Avg. money	Avg. level	Win ratio
GOAP	5.6	0.0	16.7	77.60	4.0	1.90	0
GOB	30	0	0	150	15	3	0
MCTS Biased Limited	-2.8	5.0	27.6	106.0	5.0	1.8	0
MCTS Biased	-1.7	0.0	20.4	45.4	0.0	1.0	0
MCTS	-3.8	0.0	21.6	30.2	1.0	1.0	0

8 Secret Level 3 - Comparison of the algorithms

For each of the algorithms we perfomed 10 runs(besides GOB which is deterministic) and collected statistics of Sir Uthgard.

8.1 Algorithms set up

For all MCTS variations we tried to set as high iterations number as possible without losing the fluidity of the game.

1. GOB - we used the original function for distance calculation,
2. GOAP - depth was set to 3, we used the original function for distance calculation,
3. MCTS - no. iterations per ChooseAction was 5000, no. playouts was set to 5. we used our function for distance calculation,
4. MCTS Biased - no. iterations per ChooseAction was 5000, no. playouts was set to 1, we used our function for distance calculation,
5. MCTS Biased + Limited Depth - no. iterations per ChooseAction was 5000, no. playouts was set to 1, depth limit was 10, we used our function for distance calculation.

8.2 Summary

The best algorithm occurred to be the GOB decision making. It was almost good enough to win the game. Second best were the GOAP and the MCTS Biased Limited, their scores are very similar. An interesting fact is that the plays with the GOAP significantly varied, from dying almost at the beginning to gaining level 3 and picking up some chests. The MCTS Biased Limited behaved similarly. Therefore these two algorithms are not reliable for this game, due to the inconsistency of the results. Other variations of the MCTS algorithm performed badly.