A picture containing tool

Description automatically generatedGommers Tijl

How do UtilityAI compare to GOAP for NPCs in a soulslike battle game in their setup and adding of npc actions.

Supervisor: Boury Fries

Coach: Geeroms Kasper

Graduation Work 2024-2025

Digital Arts and Entertainment

Howest.be

A close up of a card

Description automatically generated

Contents

[Abstract & Key words 3](#_Toc153482830)

[Preface 4](#_Toc153482831)

[List of Figures 5](#_Toc153482832)

[Introduction 6](#_Toc153482833)

[Literature Study / Theoretical Framework 8](#_Toc153482834)

[Research 12](#_Toc153482835)

[1. Topic 1 **Error! Bookmark not defined.**](#_Toc153482836)

[1.1. Subtopic 1 **Error! Bookmark not defined.**](#_Toc153482837)

[1.1.1. Subtopic **Error! Bookmark not defined.**](#_Toc153482838)

[2. Topic 2 **Error! Bookmark not defined.**](#_Toc153482839)

[1.2. Subtopic 1 13](#_Toc153482840)

[1.2.1. Subtopic **Error! Bookmark not defined.**](#_Toc153482841)

[Discussion 16](#_Toc153482850)

[Conclusion 17](#_Toc153482851)

[Future work 18](#_Toc153482852)

[Critical Reflection 19](#_Toc153482853)

[References 20](#_Toc153482854)

[Acknowledgements 21](#_Toc153482855)

[Appendices 22](#_Toc153482856)

# Abstract & Key words

**An abstract explains the outline of the paper concisely (the methods, results, etc.). Maximum length of 250 words, preferably both in English and Dutch.**

This research explores the comparative effectiveness of Utility AI and Goal-Oriented Action Planning (GOAP) in managing Non-Player Character (NPC) behaviors within a soulslike battle game environment.

Utility AI, which selects actions based on dynamically calculated utility scores, offers a straightforward setup but can become complex with an increasing number of actions. This approach has been widely used in various game genres, from strategy games to simulations, due to its ability to make informed decisions based on the current game state.

In contrast, GOAP, which involves defining goals and planning sequences of actions to achieve them, presents a more intricate initial setup but excels in flexibility and dynamic adaptability. GOAP has been instrumental in creating more dynamic and responsive NPC behaviors in games, allowing characters to evaluate their current state and choose actions that best align with their goals.

This study evaluates both approaches in terms of setup complexity, ease of adding new NPC actions, flexibility, performance, and long-term maintenance. The findings suggest that while Utility AI may be more performant in simpler scenarios, GOAP's structured planning and adaptability make it more suitable for the complex, dynamic nature of soulslike games. This research contributes to the understanding of AI methodologies in game development, providing insights that can enhance NPC behavior and overall game experience.

# Preface

***A preface is a statement of the author's reasons for undertaking the work and may include personal comments that are not directly relevant to other sections of the thesis or dissertation.* No word count limit.**

I started this research out off personal interest off npc behaviour in games.

After a previous research on UtilityAI, I was curious how this compares to others, for example FSM, BT or GOAP.

I was still unfamiliar with how Goal Action Oriented Planner worked but heard good things about it, this motivated me to learn GOAP and compare it to each other so I knew what to use and when in future projects.

# List of Figures

**The list of figures lists the figures in the order in which they appear throughout the thesis. They may be numbered sequentially, or be subdivided following the chapters in which they appear.**

Figure 1: A picture showing something

Figure 2: A graph showing another thing

Figure 3.1: A tabel showing yet another thing, that appears in chapter 3.

# Introduction

**In the introduction, you write the background of your topic and discuss the observation that spurred you on to do this research project. Explain the purpose of the paper and present your research question(s) and the hypothesis at the end of this section. This section is typically a couple of pages long.**

SoulsLike Game:

These games are known to be more difficult then others, where the player has to approach carefully and look for openings and keep an eye on their stamina and health. Attacks can not be cancelled, which is why it leaves you vulnerable if you time it wrong.

I want to find out which system can handle this while having the actions:

* Quick Attack: fast but weak attack, average stamina cost.
* Hard Attack: slow but devastating, is strong against block, high stamina cost.
* Block: Put received damage to stamina instead off health, slows down movement drastically, no stamina cost to execute.
* Heal: Takes long time to execute, heals halve off its health when completed, high stamina cost.
* Throw: Send an object flying to the opponent, low damage, high stamina cost.

UtilityAI:

Decision making forms the core of any AI system. There are many different approaches to decision making. One of the most robust and powerful systems we’ve encountered is a utility-based system. The general concept of a utility-based system is that every possible action is scored at once and one of the top scoring actions is chosen. By itself, this is a very simple and straightforward approach[1].

GOAP:

By planning in real-time, we can simulate the affect of various factors on reasoning, and adapt behavior to correspond. With F.E.A.R., we demonstrated that planning in real-time is a practical solution for the current generation of games. Moving forward, planning looks to be a promising solution for modeling group and social behaviors, increasing characters’ command of language, and designing cooperative characters[2].

*How can each off them be used in this soulslike battle game, how do they compare to each other on setup, implementing each action, their reaction to the actions and their maintainability?*

*Gameplaywise, how are each in their management off stamina and reactions to the player.Comparing the time put in the actions in both systems, which one pays off the most.*

The following hypotheses were formulated:

* Null hypothesis (H0): There are no remarkable differences, they both work just as well and as easily implemented as the other.
* Working hypothesis 1 (H1): UtilityAI is easiest to setup, but hard to maintain after more actions gets implemented. GOAP will be much easier to maintain and ends up more stable at the end.

# Literature Study / Theoretical Framework

**In the literature review, you present the secondary research you have conducted. You detail the background of your topics and write about the concepts that are relevant to the study. Assume that not every reader has the same skillset or -level as you do! This section typically requires a substantial amount of references and can be a lengthy section that requires a considerable amount of pages.**

## Decisionmaking algortithm

A npc in games that attacks you, runs away or is goofing around are all different kind off behaviors he can do. What behavior he will do is decided by his decision making algorithm.

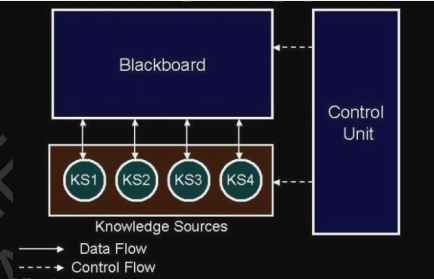
What they all have in common is that they use the world state (game environment) to decide on their next behavior. A commonly known way to get this state values is by the use off a Blackboard.

e.g. You are patrolling an area (behavior = patrol), when you suddenly notice an enemy (state enemyFound = true), so you run to the enemy (behavior = GoToEnemy), when you reach him you attack (state enemyInRange = true , behavior = AttackEnemy).

Most known examples are the Finite State Machine(FSM) or the Behavior Tree(BT), in this research we will be talking about UtilityAI and Goal Oriented Action Planner(GOAP) and find how they differ in selecting new behavior.

## Blackboard

The blackboard architecture is a flexible and powerful expert system framework. It represents a general approach to problem solving that is useful in many domains of applications especially in the area of intelligent control. The blackboard architecture can provide an environment for achieving intelligent control behaviors in many AI systems[4].



Knowledge Sources: this are independent agents in the game, e.g. player character, door off a building, plants in the scene,…

Control unit is the agent that owns the blackboard.

When an agent his state changes he notifies the blackboard, when the decisionmaking algorithm off other agents need that value he gets it from the blackboard instead off trying to look for connection with that previous agent and get his value.

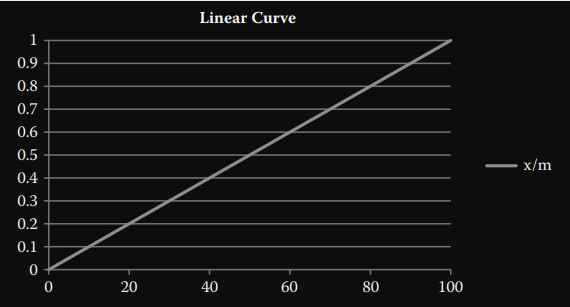
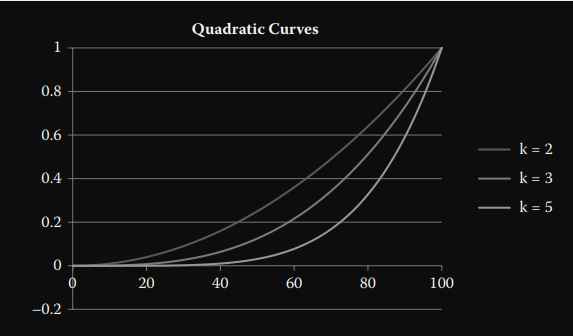
## Utility AI

* **What is it**

Utility Based AI is a method of assigning scores to  based on some heuristic, usually involving [Curves](https://grail.com.pl/documentation/documentation/1.2.0/manual/utility/curve.html) and Weights. Then algorithm analyzes those scores and chooses the most suitable [Behavior](https://grail.com.pl/documentation/documentation/1.2.0/manual/general/behavior.html).[3]

When calculating utility scores, it’s important to be consistent. Because utility scores are compared to each other to come up with a final decision, they must all be on the same scale across the entire system. As you’ll see later in this article, scores are often combined in meaningful ways to produce other scores. Therefore, using normalized scores (values that go from 0–1) provide a reasonable starting point[1].

* **Curves**:

The curves used by the algorithm determine the score the action gets compared to its world state.

The Y-axis is normalized but the X-axis can be whatever is needed.

E.g. score calculated on the distance to the opponent.

When the distance is lower then 50, he is close enough to hit Y = 1. When the distance is over 350, he is to far to matter Y = 0.

* **Weights**:

Several curves can be used to calculate the score off 1 action, but not both curves are equally important.

To adjust this, we can give weight to the curves.

The weights are a number that we multiply the Y value from the curve, and important is that we keep the total weight off all curves.

To calculate the score off this action is then by adding all the weighted Y values and then divide by the total weight.

Score = (0.5\*2 + 0.8 \*0.5)/2.5

* **Summary**

Every possible [Behavior](https://grail.com.pl/documentation/documentation/1.2.0/manual/general/behavior.html) get one or more curves with their perspective weight assigned to it. On every update the score off this behavior is calculated. The behavior with the highest score gets executed.

## GOAP

* **What is it**

Goal oriented action planning is an artificial intelligence system for agents that allows them to plan a sequence of actions to satisfy a particular goal. The particular sequence of actions depends not only on the goal but also on the current state of the world and the agent. This means that if the same goal is supplied for different agents or world states, you can get a completely different sequence of actions., which makes the AI more dynamic and realistic[5].

The most important parts off this algorithm are Goals, Actions, Planner and the WorldState.

* **Goals**

Goal is the endpoint the AI wants to achieve, this is where the plan off actions is build upon. E.g. KillPlayer, KeepHealthUp

The way the goal is selected is by priority, the goals are ranked by priority in a list and the first valid one will be selected.

The is valid state means if the goal is worth to be pursued [6].

* **WorldState**

WorldState is the state off the game, the collection off all the important worldvariables needed for the algorithm.

The current WorldState is a representation off how the game currently is, this can be a list off bools or enums.

Every goal has a desired WorldState, this means that for the goal to be achieved the current WorldState has to be equal to the desired WorldState.

* **Action**

An action is something that the agent does. Usually it is just playing an animation and a sound, and changing a little bit of state (for instance, adding firewood). Opening a door is a different action (and animation) than picking up a pencil. An action is encapsulated, and should not have to worry about what the other actions are.

To help GOAP determine what actions we want to use, each action is given a cost. A high cost action will not be chosen over a lower cost action. When we sequence the actions together, we add up the costs and then choose the sequence with the lowest cost.[5]

Actions have a Satisfy WorldState and can have a Desired WorldState just like the Goals.

* **Planner**

The planner will start from the selected goal and will look at his desired worldState.

For every state in this collection that has to be achieved, he will look for an action to achieve this.

The planner will go over the entire list off actions, if he finds an action that satisfies the desired state, he stores the cost, looks if that action has Desired WorldState as well and looks further until no more desired state necessary.

The planner uses A\* pathfinding to get the cheapest plan to reach all Desired WorldState.

* **A\***

The *A (A-star) pathfinding algorithm*\* is a popular and efficient method used to find the shortest path between two points on a graph, such as a map. It combines the strengths of Dijkstra's algorithm and a heuristic approach to optimize the search process[7].

**How A\* Works**

1. **Graph Representation**: The environment is represented as a graph, where each node corresponds to a point (e.g., a grid cell), and edges represent the connections between these points.
2. **Cost Functions**:
   * **g(n)**: The actual cost from the start node to the current node ( n ).
   * **h(n)**: The heuristic estimate of the cost from the current node ( n ) to the goal node. This is often calculated using methods like the Euclidean distance or Manhattan distance.
   * **f(n)**: The total estimated cost of the cheapest solution through node ( n ), calculated as ( f(n) = g(n) + h(n) ).
3. **Algorithm Steps**:
   * **Initialization**: Start with the initial node, setting its ( g )-cost to 0 and calculating its ( f )-cost.
   * **Open and Closed Lists**: Maintain two lists:
     + **Open List**: Nodes that need to be evaluated.
     + **Closed List**: Nodes that have already been evaluated.
   * **Node Evaluation**: Select the node with the lowest ( f )-cost from the open list. If this node is the goal, the path is found.
   * **Neighbor Processing**: For each neighbor of the current node, calculate the ( g )-cost and ( f )-cost. If a better path is found, update the costs and set the current node as the parent of the neighbor.
   * **Repeat**: Move the current node to the closed list and repeat the process with the next node from the open list.

[7]

* **Summary**

GOAP will create a plan based on the first most important goal that he needs to achieve, he makes this plan by selecting all the actions needed to satisfy all the Desired WorldState using the A\* pathfinding to ensure the most proficient one.

# Research

## Experiment setup

What data do we need to find out which system is most useful for what type off project.

### Setup time

How long does it take to setup each system?

🡪This is good to know for when a developer has a big or small project in mind. With only a small one, it is not interesting to use a system with a big setup time where it will not even be used to its full strength.

### Time to implement a new action in the system

Is it a straightforward approach to make it do the action at the time you want it to?

When you add a new action to the list, will it disrupt the previous action their execution?

### behavior in game

Will the fully implemented system act as you programmed it to do?

🡪When you implement the system to act in a certain way, it is off course important that this will happen. E.g. When set it that the AI will heal itself, you don’t want him to rush at the target to attack.

## How to get this data

### setup time

We will gather this information by implementing our own implementation off both systems in a combat game we designed for it.

🡪Log the time it takes to have a working implementation on each off them. We start with UtillityAI then GOAP.

Input variables: The research and paper prototype, only start logging when know what to do.

Output variables: The logged time.

### action implementation

Continue on previous implementation, we will create signals for specific actions to be executed by the character.

For every type off action the character can do, there has to be a at least 1 signal off them each.

This signal has to only be send at the by us designed situation.

🡪We log the time to create every signal.

🡪After every new signal is added, test if previous still work as before. If not, redo last created signal or adjust previous on.

Input variables: The strategy we want both AI’s to follow when fighting his opponent.

Output variables: The logged time.

### behavior in game

* We set the rule that the AI has to attack player while keeping his stamina usage in check.

This will be done during the creation off the signals.

We choose for stamina because this is the thing he needs for everything. If he fights with no regards for it, runs empty, he loses the ability to execute the correct action on critical times.

E.g. Heal when his health is low and far away. Disrupt the opponent healing by throwing a rock or attack him.

* The AI has to be able to win from the player.

Even if he keeps an eye on his stamina, if he cant win it wont matter.

🡪To test the 2 above behaviors, we will let both implementations be tested by a human player.

We will log both the stamina from the AI and the player and the wins/loses.

By having the stamina off the player in comparison with the AI, we can see how the player gets pressured by the AI.

We will ask 15 people. Age 15 – 40. Experienced with soulslike games as testers.

The age is chosen for their combination of reflexes and experience in games.[8].

Younger is not chosen because of their lack of patience needed for soulslike games.

Every tester will play 3 minutes against UtillityAI implementation, then 3 minutes against the GOAP.

Input variables: The game will equal in all ways for both the AI and player.

Output variables: The logged stamina off both systems and the player, the logged wins/loses.

## Software used

### unreal engine 5

Started with the 3theperson template. Made a new GameMode and controller for the player. Developed 5 BattleActions representing the 5 actions that both the player and AI need to be able to do. Made sure there is a constant *Lock on* to each other.

Create the AIController that can execute both algorithms.

Create the blackboard, his keys and update this every tick.

### SoulsCharacter

This is the character which will be controlled by the player and AI, they will both have the exact same stats and actions. It holds:

* 5 BattleActions:
* MovementComponent (inherited from AActor)
* HealthComponent
* StaminaComponent
* KnockbackComponent
* A TRingBuffer off BattleActionBase: This is the queue off actions that are selected to execute. It pops from the front and gets added in the back. The popped action gets executed. An action will be automatically removed after a set amount off time.

When an action gets executed the player is unable to execute another action, a timer will start that will enable the player again when finished.

When character gets stunned from a hit. The current action and the action Queue gets reset and will be unable to activate a new action for as long as the stun endures.

### BattleActions

They are all derived from the class BattleActionBase. The state they all need to have are:

* Cost: the cost off stamina to execute this BattleAction.
* Damage: the amount it subtracts from the character he hits.
* ExecutionTime: this is the time it takes to finish the BattleAction, while it is active the characterMovement is disabled.
* KnockTime: the amount off time the hit character will be stunned and receives a knockback. The knockback itself is handled by the KnockbackComponent off the player.

The methodes that they need to have are:

* Execute: This is the animation off the action.
* DeActivate: This is an event triggered from the baseClass to stop executing the animation.
* CollisionHandling: This is responsible for the reaction off getting hit.

When he hits the character body different things can happen

* + Was character already stunned? Nothing will happen then.
  + Was character blocking? If this was true, the damage will be dealt to the staminaBar instead off the healthBar. No knockback will be executed.
  + Was the stamina amount to low to block? Then the stamina will be zero and he will receive knockback and health damage.
  + Was there no block? Then character will receive the knockback and damage to his health.

The BattleActions themselves are created in blueprint, this is where they get their own specific state and execution.

For their to be meaning to the action/reaction off the AI, their need to be some kind off rock/paper/scissor to these BattleActions.

E.g.

Block is perfect against QuickAttack but weak against HardAttack while HardAttack is easily countered by quickAttack.

Throw is weak and costly but perfect when the opponent is far away and starts to heal.

### AI Controller

Called Npc\_Controller, the state it needs to have are:

* AIBehaviourBase m\_SelectedBehaviourSystem: Thid is the selected DecisionMaking algorithm, the brain off the Character.
* Blackboard: The only place where it gets created and updated.
* ASoulsCharacter called m\_NpcRefrence: Reference to its controlled character

On Tick, This will execute the *SelectedBehaviourSystem* , returning an EAction enum. A switch case will handle this enum to the concrete action that needs to be executed.

EAction

* Idle ,
* WalkForward ,
* WalkSideways ,
* WalkBackwards,
* Block ,
* StopBlock ,
* Heal,
* Throw ,
* QuickAttack,
* HardAttack

From then on it will work same way as for the player, e.g. pressing forward arrow equals the reaction in code as when the enum WalkForwards is returned.

The time an action is allowed to stay in the queue is for the AI (0.2s) shorter then for the player(2.0s).

### Blackboard

State:

* String Key
* Float value

Methodes:

* AddBlackboardValue
* UpdateKey
* GetKeyValue

This is created in the AIController and updated every tick there.

Keys used in both algorithms:

* IsEnemyBlocking
* EnemyDistance
* Stamina
* Health
* EnemyHealth
* IsEnemyHealing
* IsEnemyThrowing
* IsEnemyAttacking
* IsEnemyQuickAttack
* IsEnemyFrontalAttack
* RandomFloat
* IsBlocking
* IsHealing

## AI Behavior base

The base class off the DecisionMaking algorithms, holds reference to the blackboard and the virtual method Execute().

## Implement Utility ai

### SETUP

I started this implementation on 10-11-2024. From 13:27 until 16:47. Then from 19:00 until 20:02.

11-11-2024 |20:05 -21:33. Debuging. Everything was implemented but caused crashes after few runs.

13-11-2024 | 20:00 – 21:17. Continue debugging. I was able to fix the trouble. Had to do with my wrong use off unreal at the time.

15-11-2024 | 22:02 – 23:57 Worked on the AIBehaviourBase and How to read from the UnrealGraphs.

🡪Complete **setup** worked at a total off **9 hours 2 minutes**. In this time it included the blackboard key creation and AIBehaviourBase, also lost lot off time debugging when I used Unreal in a wrong way.

### State

* List off ActionScores.
  + ActionType: the EAction returnvalue.
  + List off Scores
    - Weight
    - Curve.
    - BlackboardKey: used as the X value in the curve.

### METHODES

### Calculate score

For every ActionScore his every score, his score is calculated as previously explained in LITERATURE STUDY 3.

### Execute

* Float highestScore = 0.5. Startvalue.
* EAction selectedAction = Idle.

If no Action scores higher then 0.5, Idle will be the chosen action. this is to eliminate scores when none were actually valid.

The highest scoring action his ActionType gets used to return to the AIController.

### Making Actions

We created 1 ActionScore for each EActionType, every tick it will go over all of them and decide what input to give to the AIController.

I logged the time to create them, and explain the step by step procedure on how I got the final result off my UtilityAI.

### Move Back and forth

The 2 ActionScores I started with where the MoveBackWards and MoveForwards. This was to start with giving the AI the correct distance to the player at all time.

Start 23-11-2024 from 16:30 – 18:29 and 20:18 – 23:24. Created 3 graphs for backwards movement and 2 for forwardMovement.

24-11-2024 from 15:10-16:13. His dodge failed regulary so worked more on the graphs.

Total time**: 6hours 8 min.**

He now moves back and forwards to keep his distance between 150 and 200.

Making sure this happened took a lot off iterating, writing down situations to compare results when with different weights on the graphs.

He Dodges a QuickAttack by moving backwards, after dodge he moves back between 150 and 200.

### Move Sideways

Start 24-11-2024 from 18:00-18:13. Creeated graphs so he would move sideways on random moment and/when opponent is using HardAttack.

Total time: **13min.** This was fairly straightforward. On moments when he was idle before he will now move sideways depending on the random number or when he needs to dodge.

### Throw

Start 24-11-2024 from 18:13 – 21:26. Make the AI throw when opponent tries to heal. Some off this time was used to reform the Throw BattleAction.

Total time: **3hours 13min.**

### Heal

Start 25-11-2024 from 21:04 – 22:32.

When the AI has to heal he first need to get at a safe distance. So first we had to readjust the Backward/Forward Movement ActionScores. We did this by adding the graph dealing with his health.

Total time: **1 hour 28min**.

### Block

Start 26-11-2024 from 20:02 – 21:58. When AI has full stamina, it will be safer to block the quickAttack instead off dodging. It also keeps him closer for a counter attack.

Total time: **1hour 56min**. No need to change the other graphs but we had to be sure the block will only be called at when is being attacked and when he has the stamina and when he is in reach off opponent.

### Quick attack

Start 27-11-2024 from 19:58 – 20:29. Made the AI use QuickAttack when he is close, has the stamina and when a random number is above 0.9.

The range off random number is 0-1, we choose +0.9 so that he will not endlessly use that attack but leaves it open for idle or HardAttack. Making him less pretictable.

Total time: **29min**.

### Hard Attack

Start 27-11-2024 from 20:29 – 21:30. HardAttack shines the best when he is smashing through a block off his opponent. So we focused his use on that moment.

Total time**: 1hour 1min**. We want it to focus mostly as a reaction to opponent block, but he also has to use it randomly as attack but lot less as the QuickAttack because the HardAttack will leave him open for counterattack.

### Maintainability and total time

All of his actions are really coupled to each other, if you change the weight on one it can cause the entire balance to break.

This makes adding new Actions more and more difficult unless it is pointed at a moment where he is still Idle.

Especially the back/forward movement is very sensitive, we realized that his distance to the player started to become incorrect when his health gets lower.

Setup time : **9 hours 2min**. (half off time had nothing to do with the implementation)

Actions: **14 hours 28min**.

## Implement GOAP

### SETUP

It took a week off research and paper prototyping to start this implementation because GOAP his setup is lot more complex then the previous one.

Start implementing in unreal:

6-12-2024 from

13:24 – 14:01. Created classes for Goals, actions and GOAPBehvaiour. 🡪37min

16:55-17:14. Created WorldState class. 🡪9min

17:14 - 18:17. Created the Execute function off the GOAPBehaviour. 🡪1h 3min

20:02 – 21:04. Worked on the WorldState and started the Planner methode off GOAP. 🡪1h 2min

🡪2h 51min

7-12-2024

13:05-13:34. Finished GOAP planner in code. Improved WorldState class. 🡪29min

13:34 - 15:26. Made it possible to create assets from the code in editor. 🡪1h 52min

15:26 - 16:26. Made Goal and Action With his desired State in editor for testing. 🡪1h

16:26 - 19:10. Made IsVallid method for actions in editior, using the blackboard. 🡪2h 44min

🡪2h 15min

8-12-2024

15:13 – 16:40. Finished basic use off the GOAP system in editor. 🡪1h 27min

9-12-2024

20:50 - 22:07. Made Idle and WalkAction for testing the update of an action to finished state. 🡪1h 17min

10-12-2024

21:00 – 22:42.Update off currentWorldState to be compared with Action/goal desiredState for 🡪1h 42min

Validation or heuristic calculations.

11-12-2024

21:12 – 23:09. Use the CurrentWorldState in the planner. 🡪1h 57min

18-12-2024

14:00-17:30 Try creating plan when there are more then 1 desired state. 🡪3h 30min

19:00-20:00 Planner reates best smallest possible plan keeping CurrentWorldState in mind 🡪1h

and each off there costs.

🡪4h 30min

19-12-2024

19:32- 19:54. Set a timet on every goal, put them temporary invalid when he takes to long 🡪22min

when to achieve his plan. Every goal got his RunMaxTime and TimeOutTime.

20-12-2024

18:53-19:27. Increased goal his IsVallid, check over more values instead off 1. 🡪34min

🡪The setup is now fully ready to be used to create plans for their first valid goal.

Total time: **16hours 55min.**

### Execute

On execute, the algorithm will start with checking if the There is already a Goal selected and if it is still valid. When not valid anymore he will quit and empty the m\_CurrentPlan.

if there is a plan. This is by seeing if the m\_CurrentPlan which is an TArray off GOAPAction is empty.

When empty

* Will select the first valid goal.
* Create a plan by looking at the goal his DesiredWorldState. For every Desired State
  + Compare the State with CurrentWorldState to see if some state is already satisfied.
  + Loop over all actions available in the algoritm and compare their SatisfyWorldState with the DesiredWorldState that still needs to be satisfied. When more actions are found he will pick the one with the lowest score.
    - Score: cost + heuristic. Cost is set on the action. Heuristic is the amount off state to be satisfied on that same action.
  + This will happen recursively for when the chosen action has DesiredState as well.
* The planner will make the shortest plan for each desiredState, state by state. E.g. When DesiredState is DistanceIsClose and NeedHighStamina. He will first make the plan for Distance, add it to the m\_CurrentPlan. Then make the plan for HighStamina and add that to back off m\_CurrentPlan.

When there is a plan, the m\_CurrentAction will be what is popped from the back off the m\_CurrentGoal. This will not happen if there is already an active m\_CurrentAction.

This action will be updated and returns the EAction input for the AIController. When the action is valid, this will be set to nullptr.

### Goals

### Attack opponent

The GOAPGoal *BP\_GGKillPlayer* is the only goal in list and is valid unless the player his Health <= 0.0.

The DesiredWorldState off this Goal is m\_GoalState = Fight. All Goal WorldState has only 1 DesiredState which is the top one called m\_GoalState.

There is only 1 GOAPAction that has a SatisfyWorldState equal to that. *BP\_GAQuickAttack* . This action returns the EAction::QuickAttack and has 2 DesiredWorldState

* M\_DistanceToOpponentState
* M\_ConditionState : AI his health/stamina condition.

The AI start with full health and stamina so this get satisfied by the currentWorldState.

For the distance he needs to look for an action that satisfies this. BP\_GAForward returns EAction::Forward and satisfies this state, since it is the onle action that satisfies this and he has no desiredState by himself. The plan is created.

M\_CurrentPlan = {. *BP\_GAQuickAttack* , BP\_GAForward }.

The algorithm will pop BP\_GAForward, update it and returns EAction::Forward so the AI will move toward the player this tick.

This Action has no update, The action his IsVallid method will return true when the distance is < 160. This will be checked by comparing it with the blackboard.

* No extra time logged since this was created during the setup period.

### Dodging

Created the BP\_GGRetreat goal, This goal is valid when the AI is within a distance off 300 to the player. The general goal off this is to create distance to the player or dodge his attacks.

m\_GoalState = Dodge. This state can be satisfied by

* *BP\_GABackward* . has cost = 1.0 and heuristic = 0.0. score = 1.0
* *BP\_GALeft* . has cost 0.5 and heuristic = 1.0. score = 1.5

The heuristic from BP\_GALeft is from his DesiredState m\_OpponentActionState = HardAttack.

When the opponent is using hardAttack, the currentWorldState put this to satisfied, making the heuristic score = 0.0. What makes the total score = 0.5 + 0.0.

🡪Conclusion: When opponent is using his HardAttack . He will take BP\_GALeft in the plan, else he will take BP\_GABackward.

Time :

19-12-2024| 14:15-14:54 . Moving backward 🡪 39min

20-12-2024| 19:46-20:38 Moving sideways 🡪 52min

**🡪1hour 31min**.

### Heal

Created the BP\_GGHealthManagement goal. This goal is valid when the AI his health gets lower then 0.4. The general goal is to put himself to safe distance and heal.

m\_GoalState = Heal. This state can be satisfied by

* BP\_GAHeal. EAction::heal. Has 1 DesiredWorldState m\_DistanceToOpponent = HealDistance.
  + BP\_GABackwardToHeal.this goal will be satisfied when distance > 650.

M\_CurrentPlan = { BP\_GAHeal, BP\_GABackwardToHeal}.

Time :

20-12-2024| 18:20 – 18:53 . Moving backward 🡪 33min

### More Attacks

Starting from the BP\_GGKillPlayer from earlier, I added more attackoptions.

* BP\_GATaunt: returns EAction::Idle, Desired WorldState m\_OpponentDistanceState = close.

Will go to face the opponent to taunt him to attack.

* BP\_GACombo: returns EAction::HardAttack, Desired WorldState m\_ActionState = QuickAttack.

Will go to player and execute a quick attack directly followed by a HardAttack.

The way this goal selects is by taking the one with the lowest score. The heauristic is equal with BP\_GATaunt and BP\_GACombo. Their cost is a random number between 0.5 and 3. They will get a new value after every new plan. BP\_GAQuickAttack his cost stay at 1.0.

Time :

19-12-2024 | 20:30- 21:29. Taunt and combo assets created. 🡪59min.

### Block

On the BP\_GGRetreat goal, it will dodge attacks when he is in range off player. So to have him dodge the throwing rocks, we added a isVallid state to the goal *isOpponentThrowing* . Now this goal will still be valid when distance>650.

BP\_GABlock: cost = 0.25, heuristic = 1.0. returns EAction::Block.

When opponent is using throw, heuristic will become 0.0, making this the cheapest action for the plan.

Time :

21-12-2024 | 15:42 – 16:01. Blockasset created. 🡪19min.

### StopBlock

When he starts a block he wont stop blocking unless he starts an attack, tis is very annoying because he moves really slowly when blocking.

I decided to create a new goal BP\_GGStopBlock. Te only thing this does is stop blocking when this goal is valid. His priority is just after BP\_GGRetreat and before the BP\_GGKillPlayer.

Time :

21-12-2024| 16:19 – 17:03. StopBlock goal and action assets created. 🡪 44min.

This took longer then expected because I changed my mind several times on how to do this.

### Throw

When the opponent decides to heal himself when he somehow was able to get far away from the AI, we need to throw a rock at him.

BP\_GGInteruptHealing: this goal has the highest priority, its only purpose is to throw a rock when the opponent starts healing when The AI wasn’t able to catch him in his attacks.

Time:

20-12-2024 | 20:38 – 20:55. Made goal and his action assets. 🡪 17min.

### Maintainability and total time

The setup takes some time but it gets easier to farther you get, there are many ways on how you can create this and I know I can still keep working to improve it. But for this research it achieves what he needs to be able to do.

For new goals/actions, it only got easier the more I started to make. I can keep adding new actions without fear for it to break.

The only thing that can cause trouble in the long run is its fixed priority for the goals. It makes it easy to make sure the algorithm will do what is necessary but sometimes a priority can change depending off situation.

Time:

Setup: **16hours 55min**.

Goals/actions : **4hours 23min .**

## Implement Player experiment

### experiment

We put in a system that logs the stamina from the player and the AI every tick as a csv file.

After every dead, in the GameMode it will keep track off every win and loss and log it after the 3 min fight csv file.

When attaining the results, the 4 stamina values will be put on graphs stamina on time.

After receiving the files from every tester, we will make an average of the stamina usage of both systems and put those off the player in the same graph.

We will also take the average off the wins/losses.

Compare the 2 averaged graphs of AI/Player with their win/Loss.

## Results

### Setup

### Actions

### Testers

# Discussion

**In this section, you offer an interpretation of the results you obtained and try to relate them to the theoretical framework you presented. This is typically not a very long section, but obviously one of the most important ones.**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac quam nec arcu semper dignissim. Nulla quam magna, varius sit amet pharetra et, dictum quis elit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Proin ullamcorper, ipsum sit amet scelerisque rhoncus, leo quam rhoncus elit, sit amet ullamcorper tellus nisi eget sapien. Suspendisse potenti. Ut non justo viverra, tempus felis vitae, elementum mi. Morbi at dui sed lacus fringilla condimentum. Duis non odio ac arcu volutpat vehicula eu et turpis.

Praesent vitae magna ante. Nulla in orci lacus. Donec quis vestibulum mi. Sed ipsum sapien, pretium maximus purus sed, bibendum consequat lectus. Aliquam porttitor dolor eu gravida vulputate. Vestibulum ut urna eget massa tincidunt ultricies. Morbi hendrerit sapien at diam tincidunt semper. Aliquam ut quam dictum quam maximus tempor sed at felis.

# Conclusion

**In this section, you ascertain the demonstrable outcomes of your study and outline the merits of the project for the academic field and the discourse community. This is typically not a very long section, but obviously also one of the more important ones.**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac quam nec arcu semper dignissim. Nulla quam magna, varius sit amet pharetra et, dictum quis elit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Proin ullamcorper, ipsum sit amet scelerisque rhoncus, leo quam rhoncus elit, sit amet ullamcorper tellus nisi eget sapien. Suspendisse potenti. Ut non justo viverra, tempus felis vitae, elementum mi. Morbi at dui sed lacus fringilla condimentum. Duis non odio ac arcu volutpat vehicula eu et turpis.

Praesent vitae magna ante. Nulla in orci lacus. Donec quis vestibulum mi. Sed ipsum sapien, pretium maximus purus sed, bibendum consequat lectus. Aliquam porttitor dolor eu gravida vulputate. Vestibulum ut urna eget massa tincidunt ultricies. Morbi hendrerit sapien at diam tincidunt semper. Aliquam ut quam dictum quam maximus tempor sed at felis.

# Future work

**This section is sometimes standalone, sometimes incorporated in the conclusion. It looks at the shortcomings of the study, alternative strategies, and what could be the next course of action in the research field. This is typically not a very long section.**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac quam nec arcu semper dignissim. Nulla quam magna, varius sit amet pharetra et, dictum quis elit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Proin ullamcorper, ipsum sit amet scelerisque rhoncus, leo quam rhoncus elit, sit amet ullamcorper tellus nisi eget sapien. Suspendisse potenti. Ut non justo viverra, tempus felis vitae, elementum mi. Morbi at dui sed lacus fringilla condimentum. Duis non odio ac arcu volutpat vehicula eu et turpis.

Praesent vitae magna ante. Nulla in orci lacus. Donec quis vestibulum mi. Sed ipsum sapien, pretium maximus purus sed, bibendum consequat lectus. Aliquam porttitor dolor eu gravida vulputate. Vestibulum ut urna eget massa tincidunt ultricies. Morbi hendrerit sapien at diam tincidunt semper. Aliquam ut quam dictum quam maximus tempor sed at felis.

# Critical Reflection

**This section is typically associated with a bachelor paper, not other forms of serious writing. It allows the student to reflect on the learning outcomes, both academically and in terms of personal growth.**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac quam nec arcu semper dignissim. Nulla quam magna, varius sit amet pharetra et, dictum quis elit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Proin ullamcorper, ipsum sit amet scelerisque rhoncus, leo quam rhoncus elit, sit amet ullamcorper tellus nisi eget sapien. Suspendisse potenti. Ut non justo viverra, tempus felis vitae, elementum mi. Morbi at dui sed lacus fringilla condimentum. Duis non odio ac arcu volutpat vehicula eu et turpis.

Praesent vitae magna ante. Nulla in orci lacus. Donec quis vestibulum mi. Sed ipsum sapien, pretium maximus purus sed, bibendum consequat lectus. Aliquam porttitor dolor eu gravida vulputate. Vestibulum ut urna eget massa tincidunt ultricies. Morbi hendrerit sapien at diam tincidunt semper. Aliquam ut quam dictum quam maximus tempor sed at felis.

# References

**In this section, you list all the references you made in alphabetical order; consequently adhere to the referencing style you have chosen.**

[1] David “Rez” Graham (14/09/2015) An Introduction to Utility Theory. <https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf>

[2] Jeff Orkin (08/10/2008) Three States and a Plan: The A.I. of F.E.A.R.

[3]  Antora default UI() Utility Based AI https://grail.com.pl/documentation/documentation/1.2.0/manual/utility/utility.html

[4] Grantham Kwok-Hung Pang (27/08/2011) BLACKBOARD ARCHITECTURE FOR INTELLIGENT CONTROL

[5] [**Brent Owens**](https://tutsplus.com/authors/brent-owens) (23/04/2014) Goal Oriented Action Planning for a Smarter AI

[6] GDC (9 Oct 2017) Goal-Oriented Action Planning: Ten Years of AI Programminghttps://www.youtube.com/watch?v=gm7K68663rA&list=LL&index=7&t=443s

[7] A\* pathfinding Copilot

[8] Mona Erfani1 , Magy Seif El-Nasr1 , David Milam1 , Bardia Aghabeigi1 , Beth Aileen Lameman1 , Bernhard E. Riecke1 , Hamid Maygoli1 , and Sang Mah2 (30/07/2010) **The Effect of Age, Gender, and Previous Gaming Experience on Game Play Performance**

https://link.springer.com/chapter/10.1007/978-3-642-15231-3\_33

# Acknowledgements

**In this section, you can thank people who contributed to your work in a meaningful way.**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac quam nec arcu semper dignissim. Nulla quam magna, varius sit amet pharetra et, dictum quis elit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin ullamcorper, ipsum sit amet scelerisque rhoncus, leo quam rhoncus elit, sit amet ullamcorper tellus nisi eget sapien. Suspendisse potenti. Ut non justo viverra, tempus felis vitae, elementum mi. Morbi at dui sed lacus fringilla condimentum. Duis non odio ac arcu volutpat vehicula eu et turpis. Praesent vitae magna ante. Nulla in orci lacus. Donec quis vestibulum mi. Sed ipsum sapien, pretium maximus purus sed, bibendum consequat lectus. Aliquam porttitor dolor eu gravida vulputate. Vestibulum ut urna eget massa tincidunt ultricies. Morbi hendrerit sapien at diam tincidunt semper. Aliquam ut quam dictum quam maximus tempor sed at felis.

# Appendices

**In many cases, there are items that were developed for a research paper that can’t go into the actual paper in full. Things suc as code, art pieces, output of statistical analysis, questionnaires, … In this section, you can present these elements; use the first page to list and number the items, then paste them sequentially. If some items are too large, you can store them online, and link to them. Common practice is to keep those links active at least one year after the publication of the thesis.**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum ac quam nec arcu semper dignissim. Nulla quam magna, varius sit amet pharetra et, dictum quis elit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin ullamcorper, ipsum sit amet scelerisque rhoncus, leo quam rhoncus elit, sit amet ullamcorper tellus nisi eget sapien. Suspendisse potenti. Ut non justo viverra, tempus felis vitae, elementum mi. Morbi at dui sed lacus fringilla condimentum. Duis non odio ac arcu volutpat vehicula eu et turpis. Praesent vitae magna ante. Nulla in orci lacus. Donec quis vestibulum mi. Sed ipsum sapien, pretium maximus purus sed, bibendum consequat lectus. Aliquam porttitor dolor eu gravida vulputate. Vestibulum ut urna eget massa tincidunt ultricies. Morbi hendrerit sapien at diam tincidunt semper. Aliquam ut quam dictum quam maximus tempor sed at felis.