Lassi Haapanen

The Environment Query System and the Spatial Query System used for AI -agency in games, a comparison

Master's Thesis in Information Technology

January 14, 2021

University of Jyväskylä

Faculty of Information Technology

Author: Lassi Haapanen

Contact information: lazzikka@gmail.com

Supervisor: Ilkka Pölönen

Title: The Environment Query System and the Spatial Query System used for AI -agency in games, a comparison

Työn nimi: Vertailu pelien tekoälyjen käytöstä ohjaavien Environment Query Systemin ja Spatial Query Systemin välillä

Project: Master's Thesis

Study line: Games and gamification

Page count: 56+0

Abstract: This study compared two systems used in game artificial intelligence that direct the interaction of AI agents and the game world around them. Based on the evaluation criteria the more advanced system capable of meeting the needs of modern development was determined. Based on the literature review we made, scientific literature on the topic seems scarce. The first of the systems is EQS, the Environment Query System which is part of the Unreal Engine 4 game engine. The second platform is SQS or Spatial Query System, part of the Kythera AI middleware. SQS was found to be the system that is a more advanced and meaningful option for a developer by its design ideals. It is hoped that the results and rationale behind the comparison will benefit both game developers choosing a game engine for their project as well as those behind the development such systems.

Keywords: games, artificial intelligence, Environment Query System, Spatial Query System

Suomenkielinen tiivistelmä: Tutkimuksessa verrattiin kahta tekoälyn tukena käytettävää järjestelmää, joita käytetään ympäristön ja tekoälyagentin välisen vuorovaikutuksen ohjaamisessa, jotta saataisiin tietää kumpi on vertauskriteerien pohjalta kehittyneempi ja moderneihin tarpeisiin vastaava järjestelmä. Aiheesta on löydettävissä hyvin vähän tieteellisiä tekstejä. Ensimmäinen järjestelmistä on EQS, Environment Query System, joka kuuluu Unreal Engine 4

-pelimoottoriin. Toinen järjestelmä on SQS, Spatial Query System, joka on osa Kythera AI-pelimoottoria. SQS havaittiin kehittyneemmäksi ja suunnitteluaatteiltaan kehittäjälle mielekkäämmäksi järjestelmäksi. Vertailun tuloksien ja tulosten perusteluiden toivotaan hyödyttävän niin pelinkehittäjiä, jotka valitsevat pelimoottoria projektilleen, kuin tällaisten järjestelmien kehittäjiäkin.

Avainsanat: pelit, tekoäly, Environment Query System, Spatial Query System

Preface

For my master's thesis I pursued a topic in artificial intelligence used in games, as my candidate's thesis involved the field of game AI as well. I wanted to do something meaningful in a real environment, so I looked for companies that are doing pioneering work in the field. Reaching out to the Kythera AI team was the right choice, and I am thankful for the support and opportunity they've given me. I thank Matthew Jack for his guidance and interesting discussions and Fabio Anderegg and Lucas Musial for their help amidst busy work days as well as the rest of the Kythera AI team who politely assisted me with my questions and endeavour. I am grateful to my family for their support and reflection. My thanks to Antti-Juhani Kaijanaho for providing a very helpful LaTeX-thesis template. Finally, my thanks to Ilkka Pölönen who gave me valuable feedback and supervised my work on this thesis.

As a hobbyist game developer I have been interested in game AI and the systems governing it for some time. My candidate's thesis is about AI implementations used in strategy games. I am especially interested in AI agents that are believable, immersive and human-like. I am also interested in the subject as a gamer and esports competitor, and enthusiastic about games and AI becoming progressively complex and lifelike. Learning about the systems that govern AI is a necessary step in gaining deeper understanding of how virtual entities and worlds are simulated, and the convincing and purposeful behaviour of AI agents. Asking these research questions is something that most game developers might ask themselves when thinking of which system to use for their project. Furthermore, taking part in and observing the industry's pioneering development and study is a pleasure in itself. I am grateful for the opportunity.

Jyväskylä, January 14, 2021

The Author

Glossary

EQS	The Environment Query System (EQS) is a feature within the Artificial Intelligence system in Unreal Engine 4 (UE4) that is used to query the environment for data.
SQS	The Spatial Query System (SQS), a part of the Kythera AI middleware, is a powerful general-purpose API to describe and efficiently process location-based queries.
Artificial Intelligence	In the context of games, this term is seen used more broadly to refer to any sort of algorithm or system that defines the agency of an entity that interacts with the virtual world around it. Some might call the scripting of a piece of rock that falls to block the player's passage when approached AI, while some think more of a complex entity learning, reacting to and countering the player's strategic moves on a geographical map. In these cases, each entity requires some sort of intelligence to define its agency. In the case of the rock it can be a primitive state machine, while a sophisticated neural network might adapt to a player's playstyle in a strategy game. In modern discourse "AI" often refers to the more complex intelligence behind agents rather than simple scripting.
FPS	First-person shooter, a game usually viewed from the eyes of the character the player is controlling.
Agent	A character or entity in a game that is controlled by the AI, and has some sort of agency. A complementary definition from Wikipedia's article, 'Intelligent Agent': "In artificial intelligence, an intelligent agent (IA) refers to an autonomous entity which acts, directing its activity towards achieving goals (i.e. it is an agent), upon an environment using observation through sensors and consequent actuators (i.e. it is intelligent)"
NPC	Non-playable character, usually an AI-controlled entity in the

	game.
Raycast	A ray cast from one point to the other to determine if there is an obstacle or object on the way that the ray intersects with.

List of Figures

List of Tables

Contents

1 Introduction

For those who work in the game industry or develop systems related to game AI the question of which engine to use in a project is also becoming more and more about what kind of benefits and assets it provides concerning the development of artificial intelligence. This thesis aims to provide answers for that along with associated criteria and reasoning. The reader will hopefully also receive new insight about evaluating engines by their query systems and understand more closely the benefits and rationale behind using an advanced query language. This thesis was partly inspired by input and discussion with the Kythera AI team (https://www.kythera.ai/) and Moon Collider, the company behind Kythera AI, an industry-pioneering game engine.

1.1 Structure of the thesis

The first chapter introduces the thesis' topic and context to the reader without assumptions of previous knowledge or expertise. The motivation for studying the topic is presented, along with a literature review of what has been studied before and how these studies relate to the topic. The research questions are then presented and the methods used to research them explained after.

The second chapter explains the theoretical background of the two systems, why they exist and how they function. The challenges and demands developers face and how these systems help overcome those challenges is discussed. Introductory and technical overviews are presented in the third chapter.

In the fourth chapter we explain the setup and function of the test scenarios. Should the reader have rightful concerns about the bias of this study towards Kythera AI, guidance is given to replicate these test scenarios and repeat the evaluation for oneself. These test scenarios are used to test the developer experience and capabilities of the two systems. Four different aspects, as per qualitative analysis, are evaluated based on both a simple and complex test scenario use case. Finally, the evaluation criteria are justified at length.

The fifth chapter reports the results of the analysis and criteria scores yielded by the comparison. We present the results both in text and graphically.

The sixth chapter discusses their implications and combines the results into suggestions and thoughts about how the systems could be improved and how future systems might benefit from these results. Additionally, we suggest future research topics and reflect on what could be improved about this thesis.

The seventh chapter compactly combines what the thesis set out to do, achieved as the results and the resulting conclusions.

Please note that the use of the pronoun "we" refers solely to the author and not any affiliates. All opinions stated are those of the author only. While the main research question is about query systems we also make comments about the behaviour tree systems and UI of both development environments, as they go hand in hand with the workflow and use of the query systems.

1.2 Motivation

Research methods were chosen according to the motivator of this thesis, which is to present useful information for both Kythera AI, a company pioneering the field of AI in games, and the wider audience, including UE4's developers, on developing the next generation of AI control systems and to improve existing ones. Including both a hands-on view on the systems and a more theoretical comparison was deemed best for highlighting both practical and theoretical matters of interest. We also hope to present useful information and reasoning to take into account for readers choosing between different game engines.

1.3 Literature review

An overview on recent research into the different areas of AI in games has been made by Xia, Ye, and Abuassba (2020). It discusses general game AI and "hybrid intelligence", which according to Dellermann et al. (2019) "refers to the ability or technology to accomplish difficult and complex tasks by merging human intelligence and AI". A deeper, more technical

study on artificial and computational intelligence has been made by Yannakakis and Togelius (2015) in their paper titled "A Panorama of Artificial and Computational Intelligence in Games", who identify ten main research areas within this field: NPC behaviour learning, search and planning, player modeling, games as AI benchmarks, procedural content generation, computational narrative, believable agents, AI-assisted game design, general game artificial intelligence and AI in commercial games. They note that there are clear imbalances between the different areas when it comes to attention from developers and researchers. One such area is the development of believable agents, which are further enabled by the continuous progress of query systems such as SQS. A conference paper by Machado et al. (2019) details a query system used for debugging games, by allowing developers "to query for game events in terms of what (was the event), when (did it happen), where (did it happen), and who (was involved)" to gain, they found, better insight into what really happened during a testing scenario. A crucial material referenced multiple times in this thesis is the Game AI Pro -book, accessible at http://www.gameaipro.com/ for free. In it, many experts of the field share insight and ways they have overcome challenges and developed games.

Query systems seem to be a topic not yet delved into by research or the academia. Accessing multi-disciplinary databases such as Scopus yielded no results of scientific papers for such keywords as "Environment Query System" or "Spatial Query System", and led me to the conclusion that scientific papers even mentioning either of the two systems are rare or non-existent. Due to the scarcity of research or mentions on the subject matter, even qualitative or superficial in nature, this thesis seems a warranted and useful approach. Keywords searched from the Scopus databse to find these sources were such as: "artificial intelligence", "history", "games", "AI", "agent", "learning", "behaviour tree", "recent" and "query system". The amount of document results for some select pairs have been listed to give some guidance on the magnitude of scientific papers available.

- "Artificial intelligence AND history" yields 4638 results.
- "games AND artificial intelligence AND agent" yields 3204 results.
- "games AND learning AND artificial intelligence AND agent" yields 1201 results.
- "games AND behaviour tree AND agent" yields 32 results.
- "Spatial Query System" yields 14 results, one of which links to the Game AI Pro

-book, while others are unrelated to games.

- "Environment Query System" yields 3 results, one of which is unrelated to games, and two of which link to the Game AI Pro -book.
- A set of keywords that was hoped to give a wide selection of source material in the subject was "games AND query system" and yielded 17 results, of which most link to unrelated subjects such as malware, airline cargo or genome sequencing. Furthermore, only four of these results both relate to games and are less than five years old.
- Another set of keywords that most closely relate to the subject of this thesis, "games AND behaviour/behavior tree AND query system" yields a single result, linking to the Game AI Pro -book.

Based on these findings it was concluded that material on SQS is understandably scarce as it is a rather new system. Mentions on EQS can be found more readily yet these papers often simply mention it as a tool used in some study or project, but not in a comparison with other systems. While the shoulders of giants this thesis must stand on were not prominent in databases of scientific material, we found them elsewhere. Many articles were found by asking for source materials from Matthew Jack, CEO of Kythera AI, and by following the trail of referenced articles from there. Presentations from game-related conferences, articles, and educational material on the query-based systems can be found and have been used as material for this thesis. As most of these materials have been presented or written by experts and pioneers in the field, we consider them similarly trustworthy and solid as any traditional peer-reviewed scientific material. Some of the experts who have analyzed relating topics for example during the GDC (Game Developers Conference) presentations include Eric Johnson, Matthew Jack, Mika Vehkala, Kevin Dill, Richard Evans, Mike Lewis, Dave Mark, Brian Schwab, Alex Champandard and Philip Dunstan. A great source for these presentations is the GDC Vault (https://www.gdcvault.com/).

The presentations used as source material for this thesis delve into the artificial intelligence used in such games as Crysis 2, Hitman: Absolution, Lichdom and more. The literature and presentations mostly concern FPS -games and as such represent a rather narrow scope on the use of the systems considering that there exist various genres that employ complex AI. FPS -games however provide a good view on the subject due to the emphasis on individual agents

and agency.

1.4 Research questions

The primary research question is: "How do the two systems compare and could they be improved?" Answering this question both provides, as previously described as the motivators of this thesis, Kythera AI with useful information on how to perfect their current and next generation systems as well anyone else in the wider audience with references and material to evaluate these systems. Therefore, a secondary, or complementing research question is: "how should newer generation systems be developed to support more complicated future AI while maintaining the best features of both systems?" Systems such as these have been used before for example in Crysis 2 (Tactical Position Selection), as explained by Jack (June 2017) and Bulletstorm (Environment Tactical Querying), as discussed by Zielinski (June 2017), among many other games.

A predominantly qualitative approach has been taken since only some factors could be quantified such as hardware requirements and hardware stress caused by the two systems. An established comparison framework or method for comparing these kind of systems could not be found either so one has been defined with the help of industry professionals in order to present a more concise comparison. Moreover, as time passes and both the hardware requirements of the systems and the power offered by modern hardware change, quantitative measurements become rapidly obsolete. A qualitative approach thus is assumed for the study to remain relevant for longer. Comparing only two systems allows us to more easily focus on qualitative rather than quantitative data, of which the latter is often used to better represent the differences between individual elements of a larger group.

1.5 Research method

The research method of qualitative comparative study with normative analysis is used. As described by Routio (2007a), the comparative method is often used in the early stages of the development of a branch of science. While the topics of artificial intelligence and agent direction in games are by no means new, as shown by the literature review section the topics

of query systems and their comparison have not been widely explored. Thus, it was felt that this method of research would suit the study best. As noted by Routio (2007a) the design of the method is simple. We take objects, the SQS and EQS, which are similar in some respects but differ in others. We list aspects defined later in this chapter and then compare, present and generalize our findings to highlight the invariances and systematic structure of the systems.

More specifically defined our method of comparison is normative. While a descriptive comparison aims to describe and explain the invariances of the objects, it does not attempt to generate changes in the objects. In fact, it usually tries to avoid them. A special style of research is needed when the aim is not just to detect and explain but also to improve the present state of the object, or to help improve or develop similar objects in the future. (Routio 2007a) This method was selected as it fits the motive of providing useful information to both Kythera AI to use in their development and improvement of their system and anyone else in the field who develops, studies or uses these systems. While our method conforms to the spirit of normative comparison, we do not strictly follow all of its steps. For example, as described by the more in-depth guide by Routio (2007c) the final product of normative analysis is a proposal. We will attempt to present some guidelines on what could be improved and taken into account in further development of query systems but stricter and more specific guidelines are avoided. This is because the systems studied often contain genre-specific or otherwise tailored logic and might not benefit from such specificity. Furthermore, we assume that as the field and the systems within evolve, increasingly specific guidelines tend to become obsolete increasingly quickly.

The thesis work has been assisted by experts and pioneers in the field and because of this some tacit knowledge and attitude is expected to be bestowed on the reader via the study as well. Tacit knowledge, as explained by Routio (2007b), is a type of knowledge only acquired through experience and often not specifically worded or explained.

Both simple and more complex scenarios were built by using both EQS and SQS -systems. The use cases were compared and the two systems analyzed according to specific criteria. These criteria are flexibility, expressivity, rapid iterativity and efficiency. Additional criteria such as abstraction, readability and extensibility are not included in the scored evaluation of

this study, but are important nonetheless are defined below and mentioned alongside other qualitative comparisons. It was felt that reliably scoring abstraction would require a much larger sample size of scenarios and testing. Scoring readability convincingly would best be done via a survey or another kind of more extensive user experience study. An extensibility scoring would be more persuasive if a more extensive amount of criteria was created and added to the language, which is not feasible in the scope of this study.

As noted by Matthew Jack in our discussions, the AI agents will be constrained if the queries are constrained, which in turn leads to the AI and gameplay suffering. The expertise necessary to define the following terms was also provided by Matthew Jack.

Definition of terms:

- Flexibility — The ability to rearrange, tweak and reapply the queries for different results, and to make continuous iteration easy and quick. Contrast this to writing custom hard-coded C++ evaluation functions, which may need to be redone due to iterations, and are hard to extend, recombine and reapply.
- Expressivity — The ability to express the full complexity of what the designers wish to create, instead of being artificially limited by design to only a small set of actions.
- Rapid iterativity — The speed at which repeated changes, tweaks and new ideas can be incorporated and implemented.
- Abstraction—The ability to reapply existing criteria (keywords) in new ways.
- Readability—The intent of a query should be understood as easily as possible by developers.
- Extensibility—The ability to easily add new criteria to the language.
- Efficiency—An efficient language itself should not add overhead to the evaluation.

In the context of this study, the SQS was used as a plugin of UE4, although it could be used in other ways, for example in conjuction with Lumberyard, Amazon's game engine ("Amazon Lumberyard homepage" 2020). The reason for this is that using UE4 as the development environment controls more of the variables regarding the learning and user experience, in contrast to using two different environments. We acknowledge this also introduces some bias towards UE4, but as it is one of the most popular engines used today, we do not consider

this a hindrance or source of inaccuracy to the study.

2 Theoretical background

Computer games with artificial intelligence (AI) -agents in them must implement a way for these agents to move in and interact with the world around them. This involves processing data of entities, objects and actors to decide what, when and how to act. Finding positions within a game world is a common requirement, chiefly for agents to move to and also for many other purposes such as spawning or choosing points to attack. During development, complex demands can evolve for the choice of location, and performance can become crucial (Jack et al. August 2020). Jack (June 2017) unfurls the developer's challenge further in Game AI Pro, chapter 33: "Choosing between positions—and generating those positions to consider in the first place—is critical to the success of these games. Not only is it key to an agent's effectiveness in combat, but it also visibly communicates his role and status in that combat. More generally in games, an agent's movement helps define his personality and often much of the core gameplay." In the same book, Zielinski (June 2017) additionally emphasizes that it is a tricky task to create a service that will supply AI with all the data it needs, at low CPU time cost while being flexible and easy to use at the same time. It needs to be able to look for different things, filter them, and score them. While many systems capable of such feats exist, this thesis focuses on comparing the EQS and SQS -systems developed for Unreal Engine 4 (UE4) and Kythera AI, respectively.

As explained by Johnson (June 2017) in the third version of Game AI Pro the ability of agents to intelligently analyze the environment to pick the best locations for the next behaviour has rapidly evolved. These systems governing the positioning of AI were once limited to the evaluation of static preset markers for such behaviours as finding cover or positions to shoot from, but today dynamic generation gives developers the ability to represent a much more sophisticated and wider range of concepts. Generating these locations at runtime allows granular sampling of the environment, so that changes in dynamic or destructible environments can be adapted to. Additionally, when generating short-term directions for AI movement, rather than a single final destination, complex movement behaviours can be represented. These behaviours include such as indirect, roundabout approaches to evenly encircle a target in tandem with teammates, or artificial life algorithms such as Craig Reynold's boids

(Reynolds 1987), all of which can be done while navigating arbitrary terrain.

In earlier decades and to this day the methods for directing AI agents in games have included methods such as state machines, fuzzy logic, swarm intelligence or even evolutionary algorithms to name but a few. While the AI systems utilizing these methods in games have potentially become very complex and highly refined in their own right, less attention has been paid towards the languages or methods used during development. Indeed, most games have developed, tested and implemented these systems by writing pure code during each phase of production. However, with the advent of query systems such as TPS (CryEngine), EQS (UE4) and SQS (Kythera AI) the rapidity of development, intuitiveness and expressivity among other factors have advanced significantly. Naturally, as the power and ease of use of these tools grows, so do the complexity and capabilities of their products, the AI agents, expand. As noted by Yannakakis and Togelius (2015), there have been a plethora of ways to use AI in games in general. This is why it is crucial that new ways and tools are invented to make development as rapid and easy as possible. Additionally, games have had and continue to have a pioneering role in many applications of artificial intelligence. AI itself will be a driving and revolutionary force as we advance as a technological society.

3 Technical introduction to EQS and SQS

Activities of AI agents can range from simple routines like patrolling between points, chasing the player or guarding an area and then carrying out specific activities when hostiles are perceived, to more complex behaviour. Adding nuance to activities like these however is how AI agents can be made more believable, immersive and human. Additionally, making the developer user experience smooth and intuitive, enabling rapid development and maximizing the developer's ability to bring their imagination to life are some of the ways a query system can truly shine. To begin seeing the connections between the strengths, weaknesses and creative potential of these systems and how they arise from their technical design and ideologies, one must understand the terms and core concepts used by the systems.

3.1 Environment Query System

The Environment Query System, EQS for short, is Unreal Engine 4's AI Tools feature. Made by Epic Games Interactive, UE4's EQS is used for collecting environmental data from the virtual game world and translating it into positions or Items to direct an AI with. First, a generator is used to generate sample data, which is then processed through a variety of user-defined Tests, returning the best Item that fits the parameters of the process. ("Environment Query System", no date) For example, a group of locations could be generated around the player at regular intervals, in the shape of a circle. Then, tests could be used to filter out the positions in the circle that are as far away from the player as possible. This would result in Items on the circle's circumference.

In the the EQS workflow one first adds a generator node either from the default templates or makes a custom one in blueprint or C++. The generator produces the locations or Actors, referred to as Items, that will be actually tested and weighted. The generator is also supplied with Contexts which are a frame of reference for the various Tests and Generators. Tests are then used by the Environment Query to decide which Items from the Generator are the winning, or best options. The best options can then be fed to the rest of the engine's components to use and work with.

3.2 Spatial Query System

The description of SQS at first does not seem that different from EQS. As explained by the Kythera AI documentation, when, for example, an agent needs to select a position to move to, it can do so by executing a Spatial Query which will return the best position for a given set of criteria. The query itself is described in a specialized query language and comprises Generators to produce a set of positions to consider, Conditions for testing to exclude inappropriate positions and Scores to rank the appropriate positions in order to find the best candidate. One can also define fallback queries that are used if no position passed all the conditions of the query. A query is first constructed and then executed any number of times by the Spatial Query System. The system handles queuing, time-slicing and asynchronous processing over a number of frames and multiple queries at once. When a query is completed, it uses the Kythera AI signal mechanism to communicate back to the original caller.

One of the developer's main tools when using Kythera AI is the Inspector, which allows for the rapid iteration of behaviour trees and deep, comprehensive debugging information on everything that is going on in the editor. Spatial Queries are defined in another file for example with a simple text editor. Changes made in the Inspector or spatial queries file are immediately available for testing which supports rapid development.

4 Material and methods

Unreal Engine 4.21.2 was used for the projects containing the simple and complex EQS scenarios. The default UE4 template for a third-person shooter was selected. UE4's own documentation was used in the setup, as well as various tutorials found in Youtube by channels such as "Ryan Laley Games". The videos can be found for example by searching for "UE4 EQS Tutorial".

Unreal Engine 4.25.4 was used for the projects containing the simple and complex SQS scenarios. Kythera AI generously provided me with a current development version of their software which comes with a UE4 plugin. The software also contains example maps, behaviours, meshes and spatial queries to name some of the content more important in the context of this study. While online tutorials were neither found or used, Kythera AI provided me with extensive documentation, a Slack channel and a few online sessions with their developers for me to ask questions.

Both EQS and SQS were tested with a simple and a more complex scenario. The simple scenario is a small map with flat walls as obstacles and an AI agent that wanders around. After spotting the player the agent follows it at a distance and attempts to retain line of sight. If line of sight is broken the agent resumes wandering. The basis for the design of this simple scenario is to test a use case scenario, where a developer with little to no experience of UE4's EQS or Kythera AI's SQS sets up simple AI behaviour, navigating the UI and using mostly default settings and logic templates provided by the systems. This provides us an understanding of the time, overhead and effort needed to produce something meaningful and get in touch with the UI and workflow. In the complex scenario the AI agent moves from cover to cover while following the player. The AI agent acts as a companion character and attempts to remain covered from the direction the player is facing. The basis of the design of the more complex scenario was to test how user-friendly and powerful the tools are for creating something new that is not achieved by simply chaining together default building blocks provided by the systems. The emphasis of this testing was on the query language and creation of the queries fed to the systems. While the scenarios were not as complex as real-world applications in games commonly are, we are confident that they provide an adequate

level of familiarity with the systems for us to apply the evaluation criteria on them. This confidence was also backed up by discussion with experts, including those of the Kythera AI team.

To make it easier for the reader to replicate the scenario, screenshots have been provided of the essential settings and setup for the scenarios. Basic UE4 editor knowledge is necessary to set up the environment and fully understand the pictures, but for anyone to follow the EQS blueprint's logic it is mostly sufficient to know that the white line between components denotes the flow of execution logic. For the SQS examples, Kythera AI documentation and some familiarity with the plugin environment is required if the reader attempts to replicate the study. Not every step or detail is provided but basic tutorials provided by UE4 documentation should fill in any potential gaps in setup knowledge.

4.1 Simple scenario: a wandering AI agent attempts to retain line of sight to the player after spotting it

The scenario has two main ideas implemented in it. The first is an AI agent walking between random points selected around it and, second, after perceiving the player, following it and attempting to retain line of sight. If sight is lost for an extended time it resumes wandering. While relatively simple, the AI agent has two states to it and already provides meaningful behaviour commonly seen in games.

4.1.1 EQS Simple Scenario

For the UE4 EQS setup the guides at "Unreal Engine 4 EQS Start Page" (2020) and "Unreal Engine 4 EQS Quickstart Page" (2020) were followed and they proved a reliable and easy way to set up the scenario. Setting up was straightforward in UE4 (4.21.2) with the help of the guide. The documentation explains the basics well as the tutorial advances.

First we enabled the EQS in UE4's editor preferences. From "General" settings, in the "Experimental" sidetab, inside the "AI" section by ticking the "Environment Query System" option as shown in Figure 1.

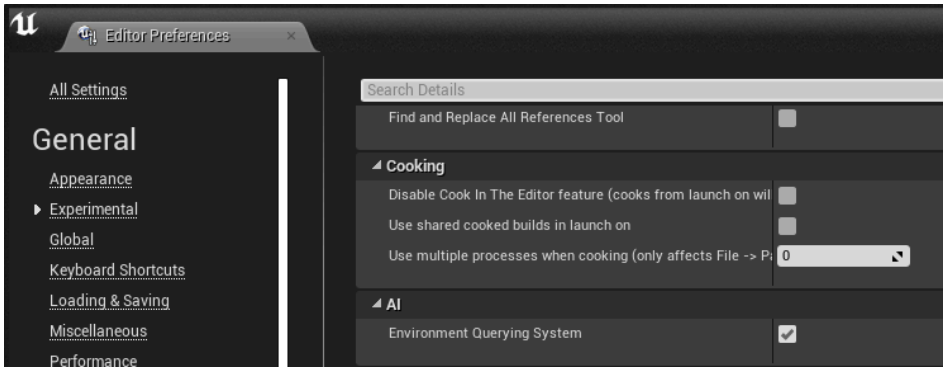


Figure 1. Setting to be ticked to enable EQS.

Next, we created the necessary objects in the folder of our choosing, named “AI” in this instance. The folder structure and files used are shown in the following Figure 2. We go through each item and explain their use next.

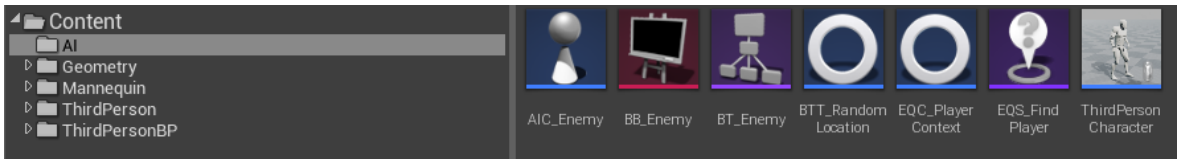


Figure 2. Folder structure and AI-folder content.

From the “Add New” menu we added a “Blueprint Class” and then created an AI Controller named “AIC_Enemy” with blueprint logic shown in Figure 3. Inside the controller two functions were created, “UpdateSightKey”, shown in Figure 4 and “UpdateTargetKey”, shown in Figure 5.

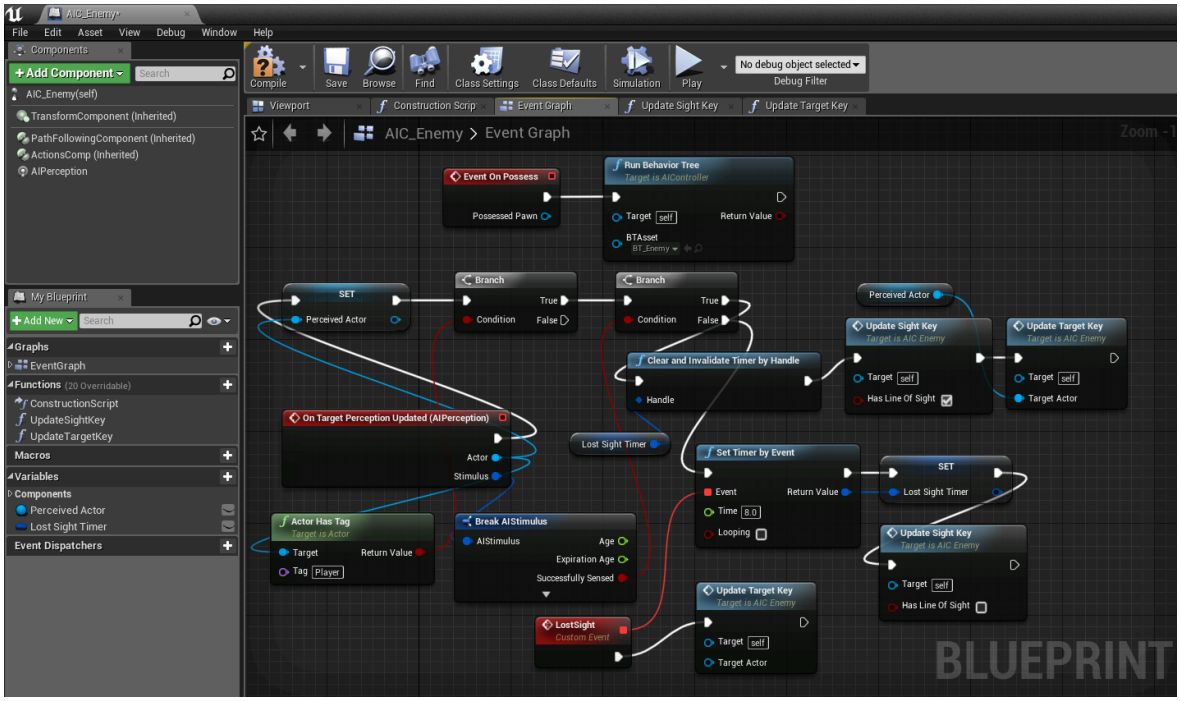


Figure 3. Blueprint logic of AIC_Enemy

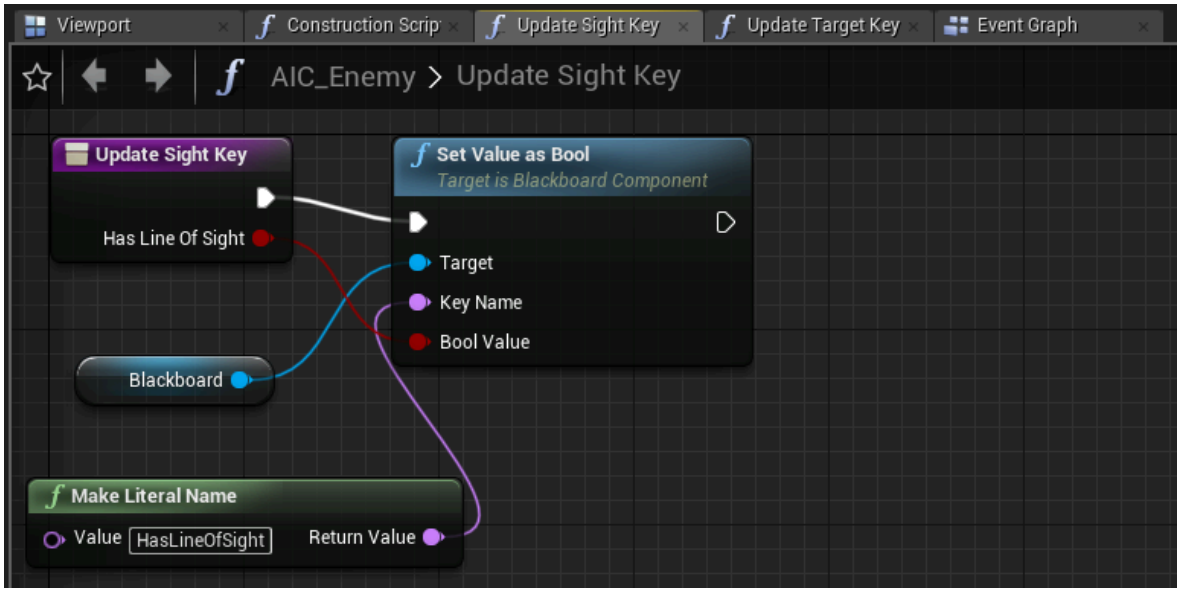


Figure 4. Blueprint logic of the Update Sight Key -function.

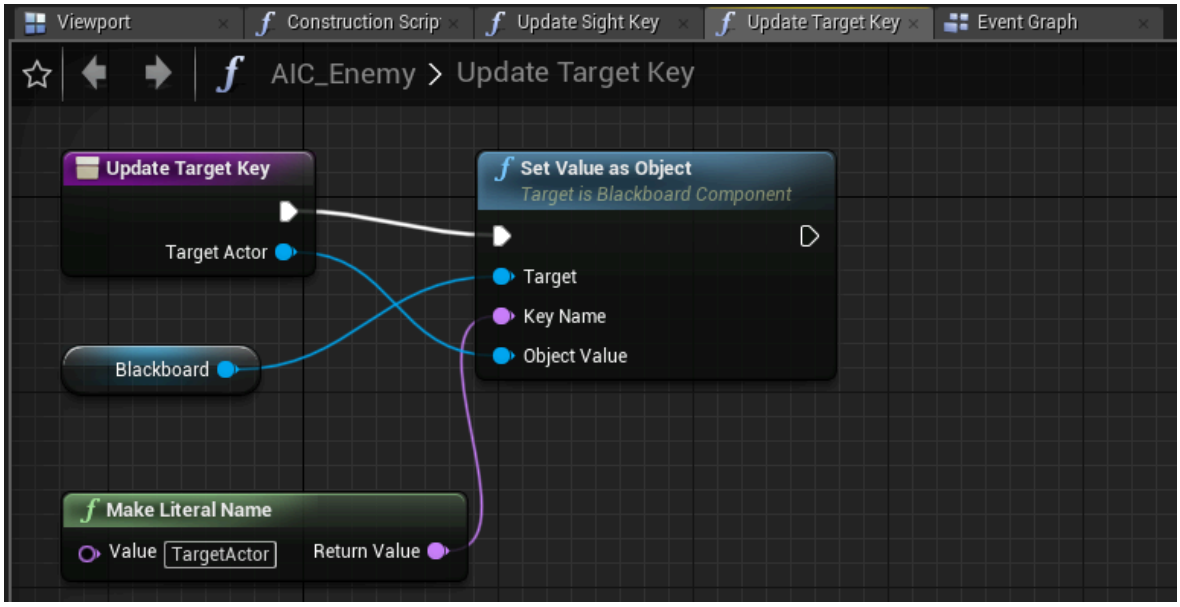


Figure 5. Blueprint logic of the Update Target Key -function.

The AI Controller specifies the behaviour tree to be ran and also what to do when another actor, in our case the player, is perceived. Perception occurs via UE4's perception system, and is based only on vision in our scenarios. The two functions simply update the necessary blackboard keys, "TargetActor", and "HasLineOfSight".

A blackboard was created and named “BB_Enemy”. Its content is shown in Figure 6.

Figure 6. Contents of BB_Enemy.

The blackboard key types are:

- “HasLineOfSight” : Boolean
- “MoveToLocation” : Vector
- “TargetActor” : Object
- “SelfActor” : Object

We then created a behaviour tree named BT_Enemy. Its content is shown in Figure 7.

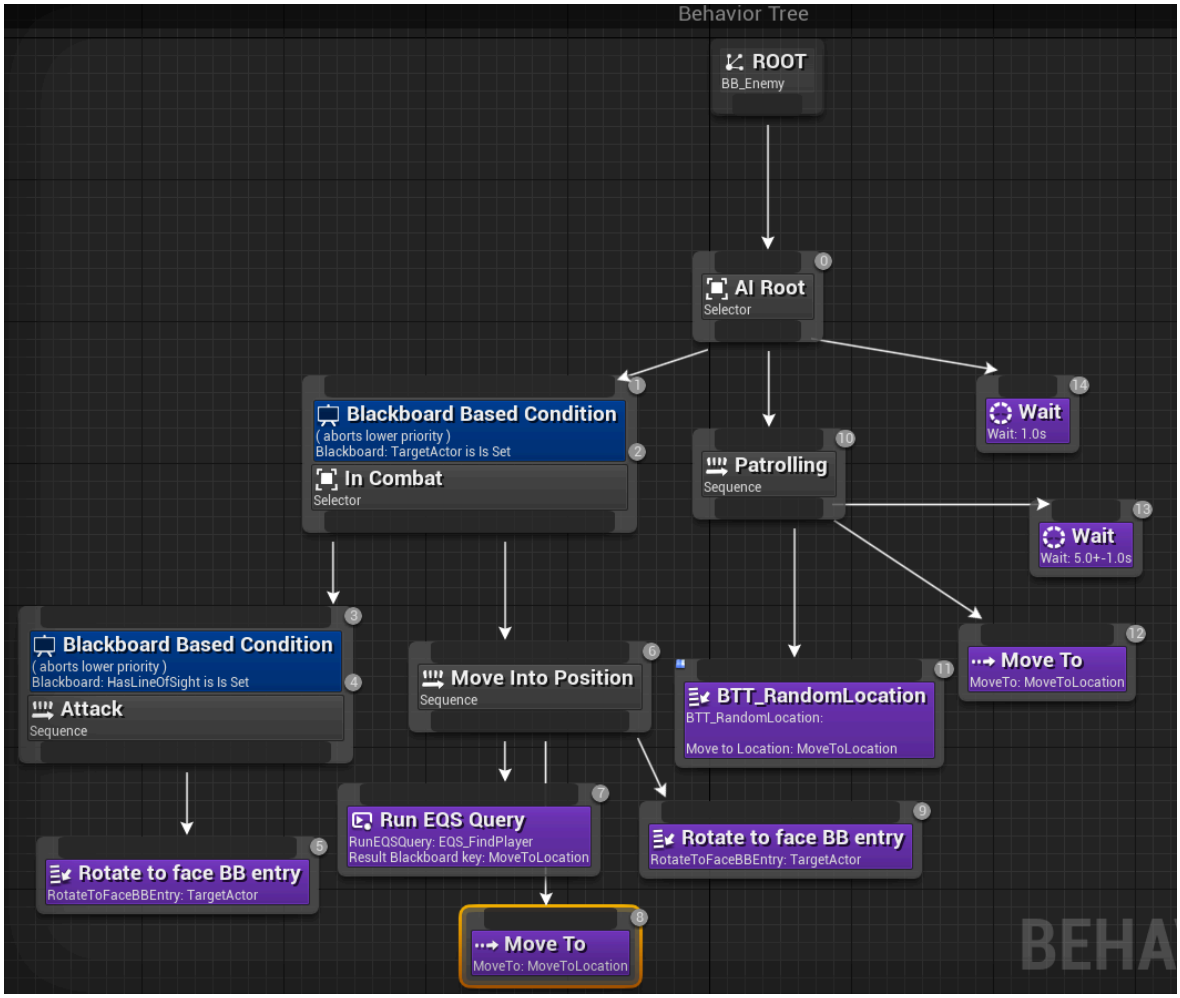


Figure 7. Blueprint logic of BT_Enemy.

Using the number at the top right corner of each node as the node numberer we specify certain settings not visible in the picture:

- Node 1 : Notify observer on result change
- Node 3 : Notify observer on value change
- Node 5 : Precision: 10
- Node 7 : Run mode: Single best item

From within the behaviour tree UI we create a new task called BTT_RandomLocation. This task will be responsible for generating a random location to patrol to when the agent is not following the player. The location is then saved in the variable “MoveToLocation”, which is visible for other blueprint components to use.

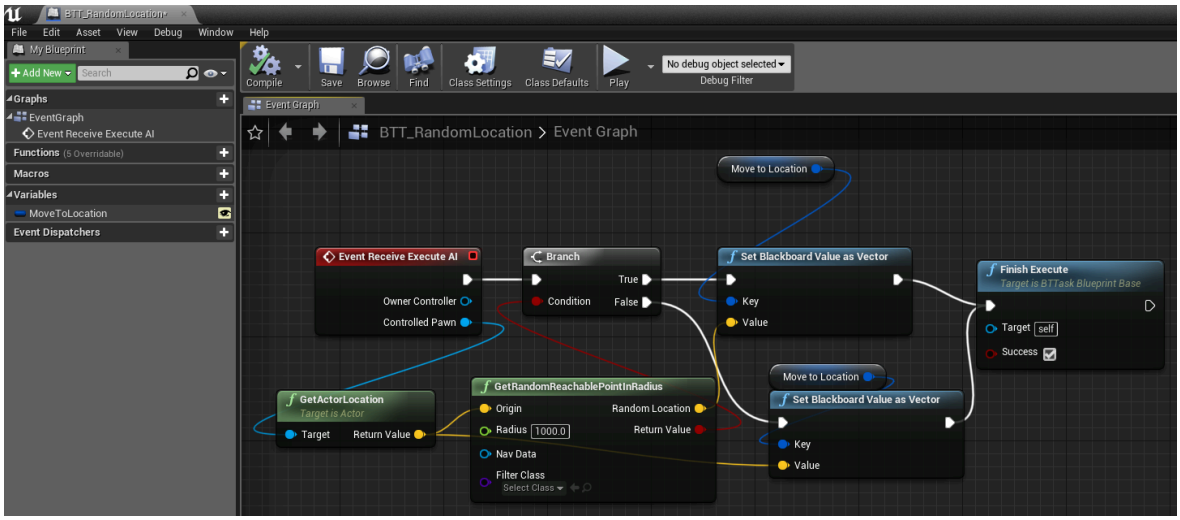


Figure 8. Blueprint logic of BTT_RandomLocation.

Next we created an environment query context called the EQC_PlayerContext. We clicked the plus-sign in the functions sidetab to override the “ProvideSingleActor”-function. Its contents are shown in Figure 9. This context is used to give EQS a reference to the player character.

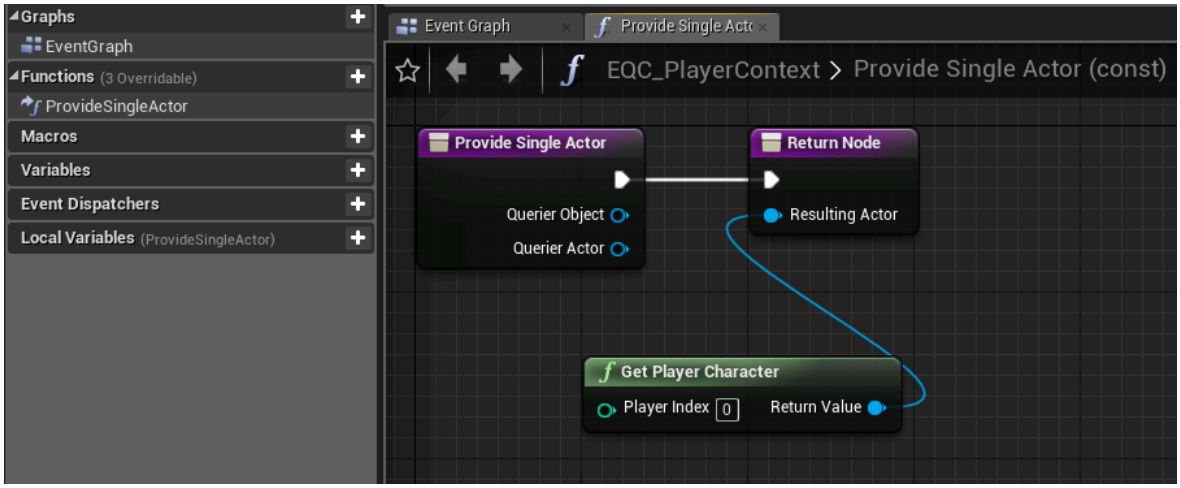
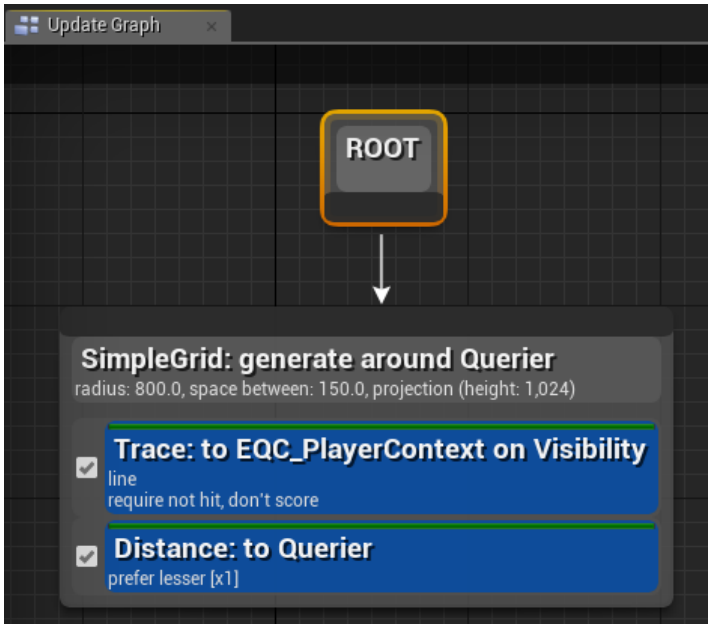


Figure 9. Blueprint logic of the “ProvideSingleActor”-function.

We then created the environment query itself as “EQS_FindPlayer”. The contents are shown in Figure 10. Settings not visible in the picture were defined and listed next to the figure.



- Trace : filter only
- Distance : score only, scoring factor = -1, scoring equation = linear.

Figure 10. Blueprint logic of EQS_FindPlayer

Lastly we dragged a ThirdPersonCharacter from the “ThirdPersonBP/Blueprints/” -folder to the AI-folder. Details of this AI agent are shown in Figure 11.

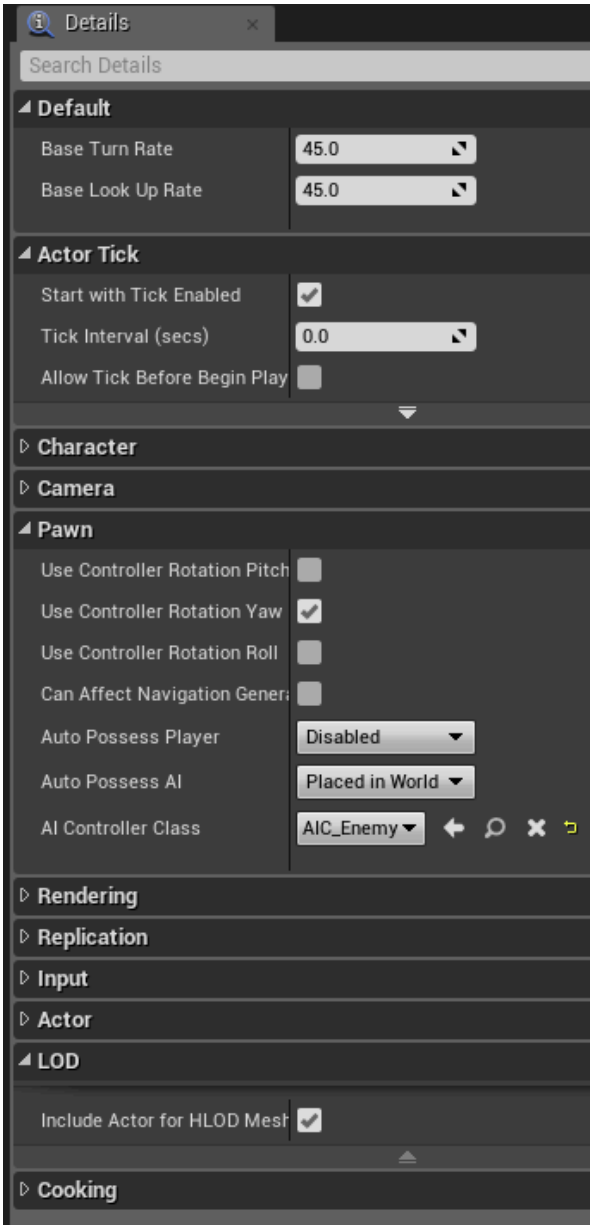


Figure 11. Details of the ThirdPersonCharacter used as the agent.

The ThirdPersonCharacter's detail panel also shows settings for the "Pawn". The AI controller must be attached here as shown in Figure 12.

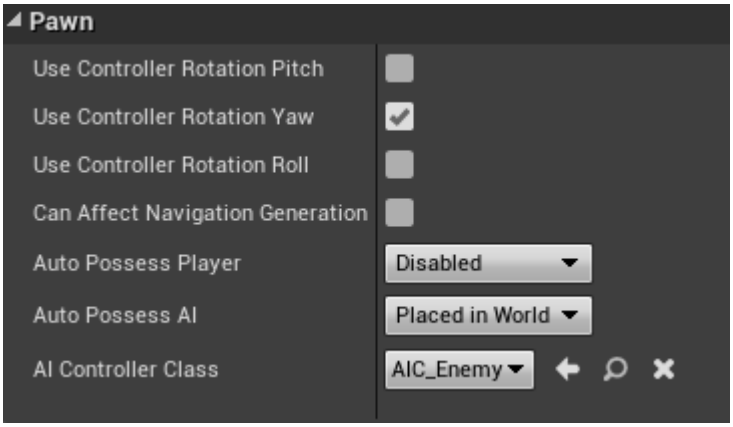


Figure 12. Details of the ThirdPersonCharacter's detailed settings in the "Pawn"-tab.

4.1.2 SQS Simple Scenario

To prepare the map an example map was used, called "AutoCoverPerceptionSignals". Flat slab-like covers were placed as shown in Figure 13. Automatically generated cover can also be seen around each obstacle. These cover-rails are not used in the simple scenario but are used in the complex scenario.

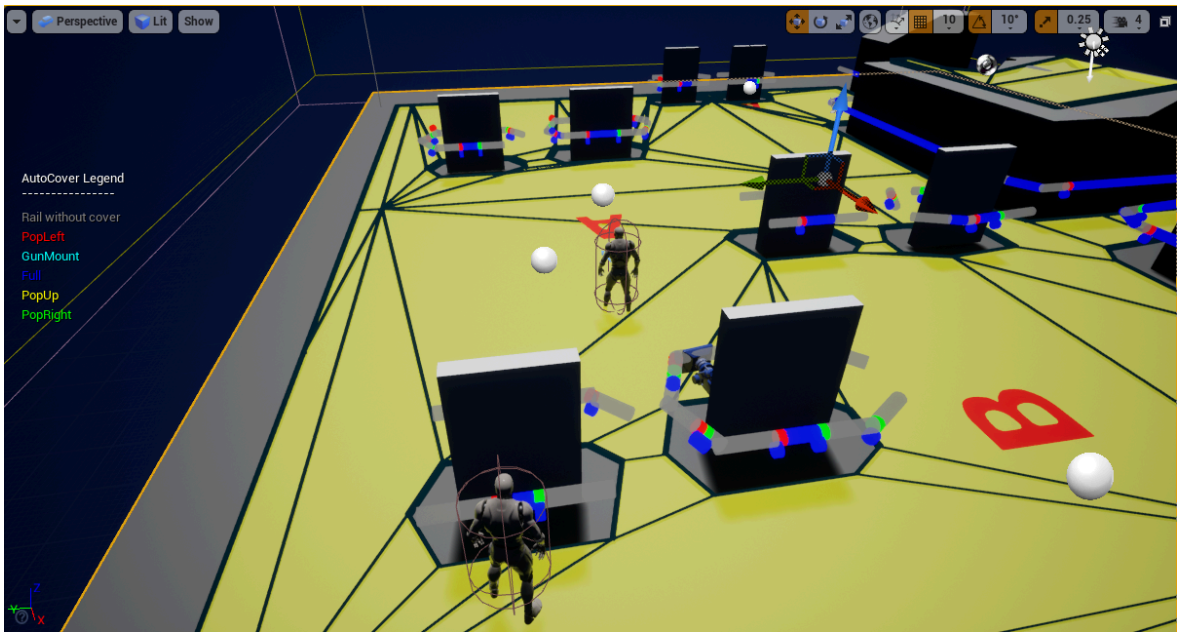


Figure 13. Viewport showing the map used for the scenario.

Using the Kythera AI documentation provided and learning from the examples of default maps and behaviours included with Kythera AI a behaviour tree and spatial query were constructed. The contents of the behaviour trees used are shown in Figure 14 and Figure 15.

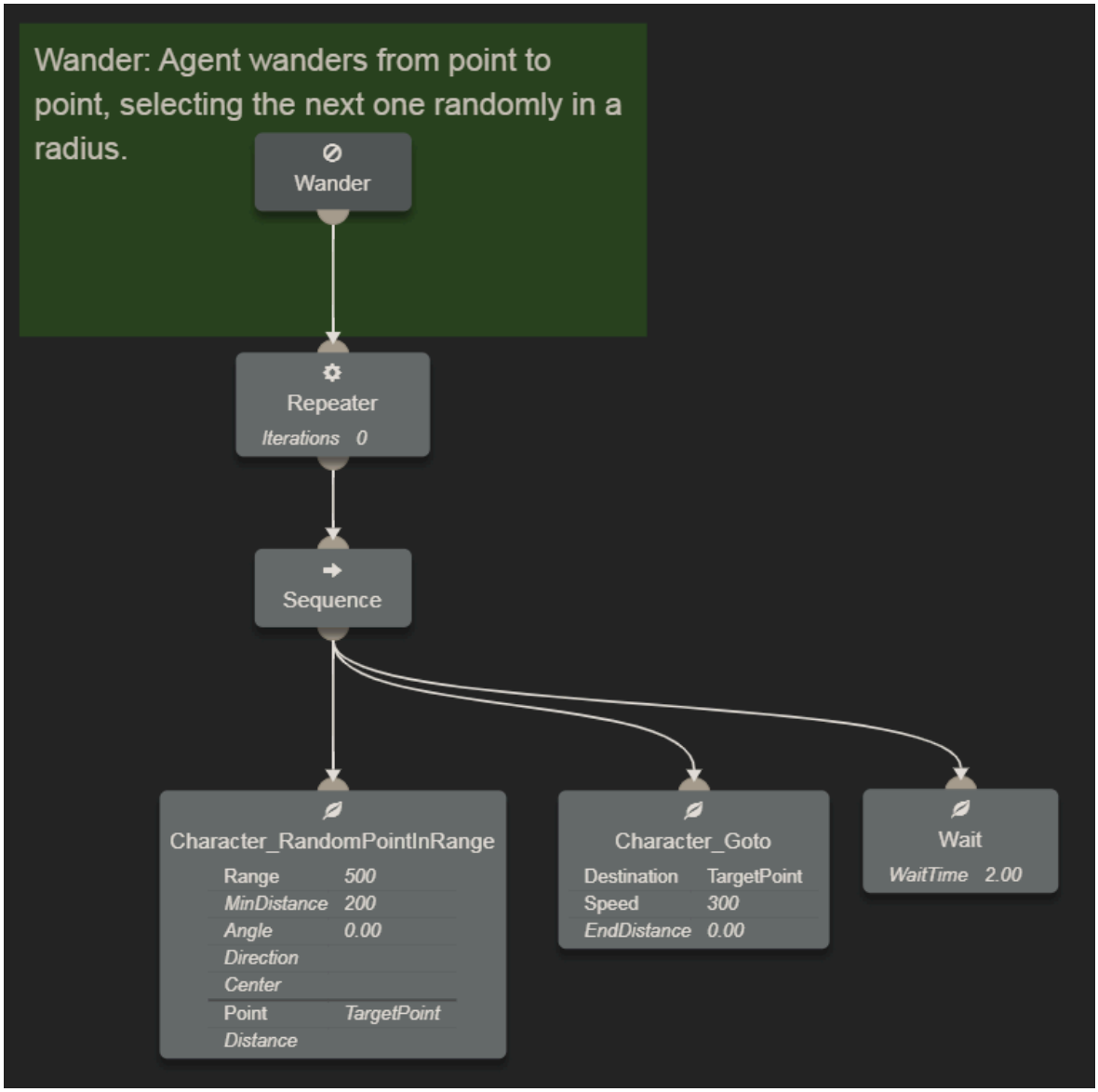


Figure 14. Behaviour tree detailing the behaviour “Wander”.

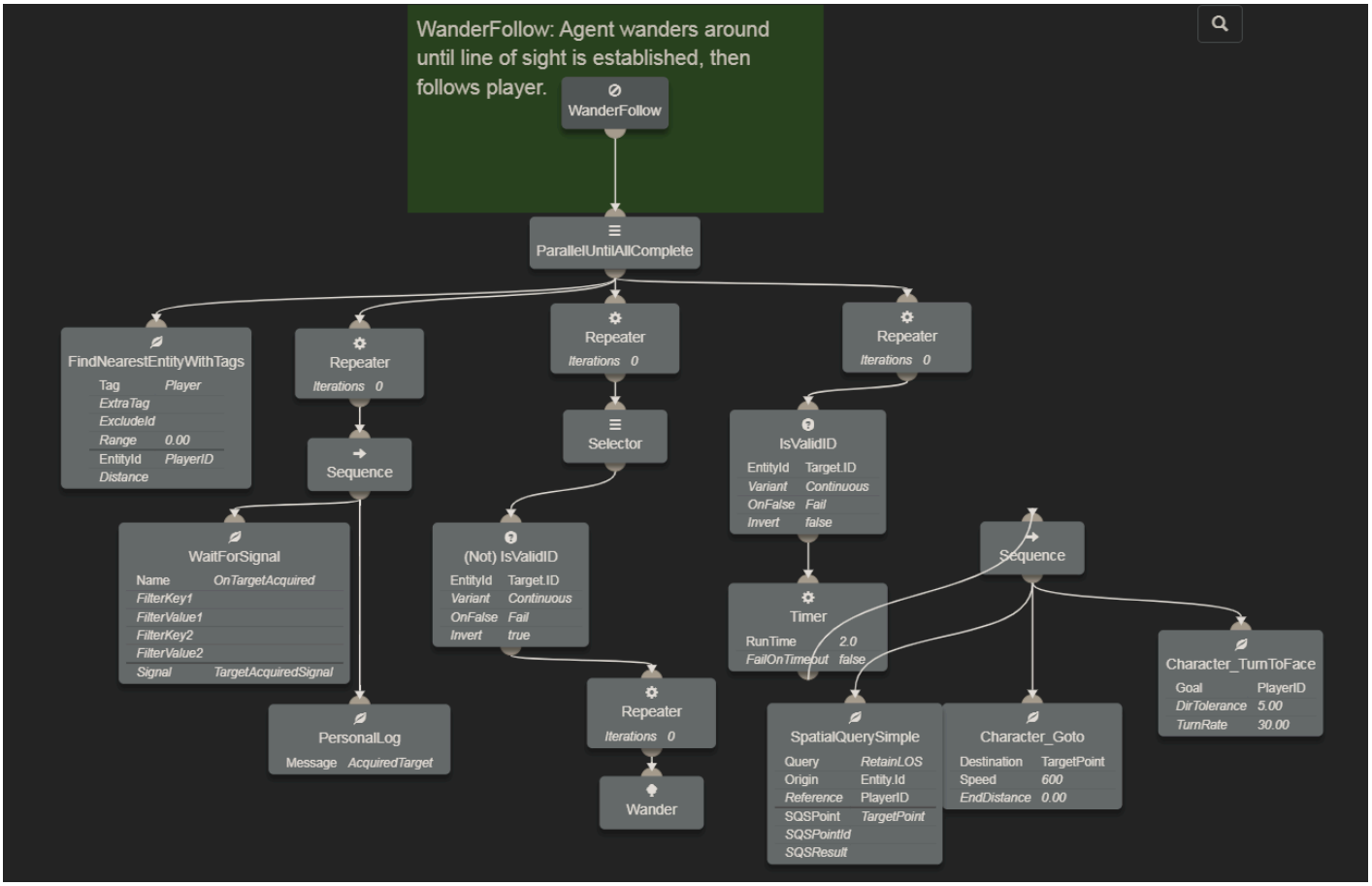


Figure 15. Behaviour tree detailing the behaviour “WanderFollow”.

The NPC was set to use the intended behaviour in the “Details” tab as shown in Figure 16.

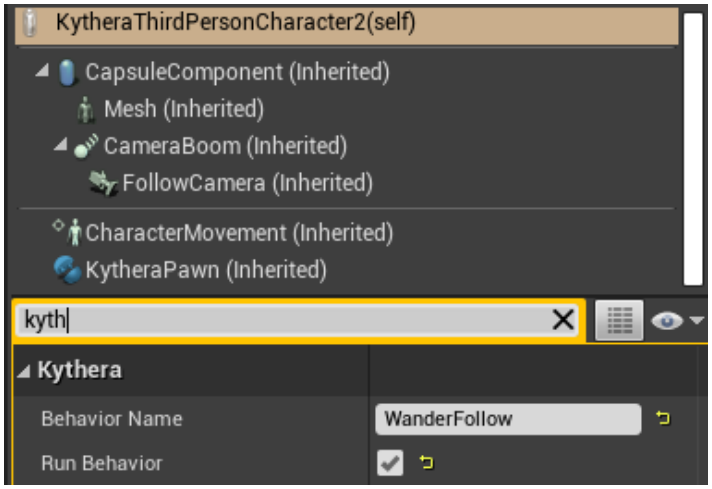


Figure 16. Behaviour “WanderFollow” assigned to NPC.

This is the spatial query used to find locations with line of sight to the player when following it. It contains a generator that creates a grid of points with the radius of the grid and space between each point as the parameters. The condition for the acceptable points are set which in this case are points with line of sight to the player. Lastly, the points are weighted so that points closer to the querier are prioritized. The condition uses a NavRaycast, but for this scenario a Raycast could have been used as both are supported by Kythera. The use of NavRaycast is simply an artifact left over from the template used for this spatial query, and while it enabled desired agent behaviour in the scenario, a Raycast would be the more correct choice in general.

```
// Query to find a spot where the player is visible from
SpatialQuery {
  Name("RetainLOS"),
  Generators{
    GridPointGenerator(SpatialQueryObject::Reference, 1600.f,300.f)
  },
  Conditions{
    NavRaycast(SpatialQueryObject::Reference, 1),
  },
  Scores{
    Weight(-0.5f, Distance3D(SpatialQueryObject::Querier))
  }
}
```

4.2 Complex scenario: An AI agent follows the player using positions covered from the direction the player faces

The complex scenario in the perspective of games in general is not that complex, but “A Somewhat More Complex Scenario” did not have the same ring to it at the time of writing this paper. The complexity it does have however comes from the AI agent’s imperative to weigh environmental factors to make decisions about its movement. The agent takes into

account both the player's location, but also locations that function as cover from the same direction the player is facing.

4.2.1 EQS Complex Scenario

Combining knowledge from various tutorials a more complex scenario was constructed. The AI agent would use environment queries to search for and weigh different positions with strong emphasis to retaining cover while following the player around. The direction the agent must remain covered from was specified by the direction the player is facing. As there are no other characters on the map, and thus no enemies or threats to reference, this was accomplished by creating an abstract threat point a specified distance away from the player, towards the direction the player is facing. Raycasts were then cast from around that point, towards that point, a set distance away. When a ray failed to reach the threat point due to an obstacle, a point on the opposing side of that obstacle from the perspective of the threat point could be then considered covered from it. These points were afterwards filtered and scored so that only those close to the player remained. The best of these points was then given for the AI agent to traverse to.

Screenshots showing the essential setup and content of the files required are shown below. It is assumed that preliminary setup of the UE4 editor environment has already been done, such as enabling of the EQS component, and setting up the ThirdPersonCharacter as described in the setup of the simple scenario.

The folder structure and files created for the project are shown in Figure 17.

Figure 17. Folder structure and files created for the project.

The types of the file objects listed in Figure 17 are as follows:

- FindCompanionCover : Environment query, found under “Artificial Intelligence” in the “Add New” -button menu.
- GenerateCompanionCover : EnvQueryGenerator_BlueprintBase, by adding a new blueprint class.
- GetIntoCompanionCover : A behaviour tree task, added from the behaviour tree editor.
- NPC : ThirdPersonCharacter, created similarly as in the simple scenario.
- NPC_AIC : AIController, by adding a new blueprint class.
- NPC_BB : Blackboard, found under “Artificial Intelligence” in the “Add New” -button menu.
- NPC_BT : Behaviour tree, found under “Artificial Intelligence” in the “Add New” -button menu.
- PlayerContext : Environment query context, by adding a new blueprint class.

The contents of the AI Controller “NPC_AI” are shown in Figure 18.

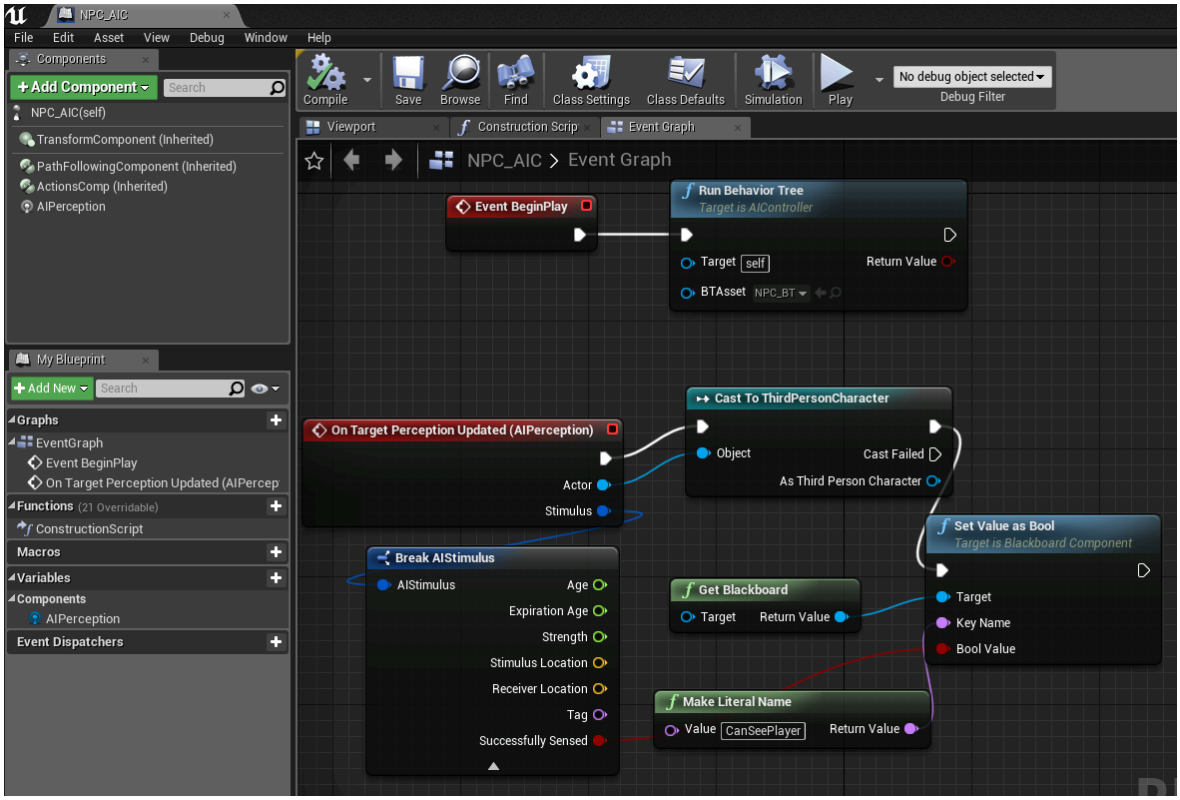


Figure 18. Blueprint logic of NPC_AIC.

The AI Controller specifies the behaviour tree being ran and also updates the variable “CanSeePlayer” based on visual stimulus.

The blackboard and its keys are shown in Figure 19.

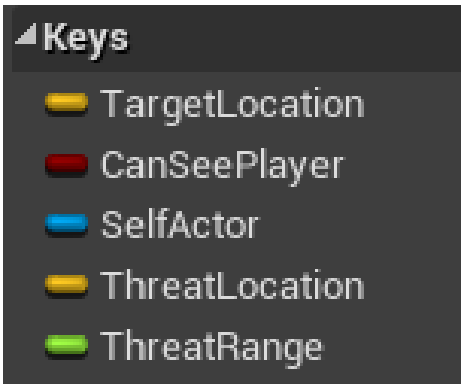


Figure 19. Contents of NPC_BB.

The blackboard key types are:

- “TargetLocation” : Vector
- “CanSeePlayer” : Boolean
- “SelfActor” : Object
- “ThreatLocation” : Vector
- “ThreatRange” : Float

The contents of the behaviour tree are shown in Figure 20.

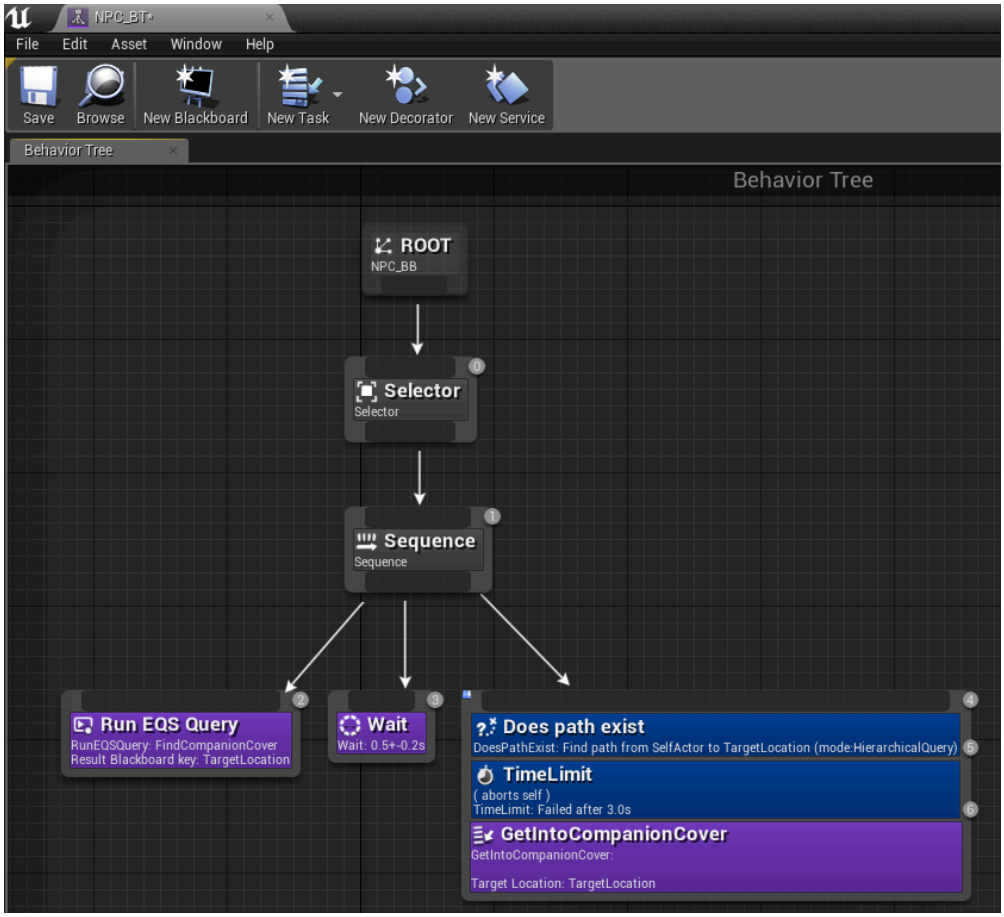


Figure 20. Blueprint logic of NPC_BT.

Using the number at the top right corner of each node as the node numberer, we specify certain settings not visible in the picture:

- Node 2: Run mode: Single best item
- Node 4: Observer aborts: none, Filter class: NavFilter_AIControllerDefault

The PlayerContext is defined as follows in Figure 21. We use the plus-sign in the functions side tab to override the “ProvideActorsSet”-function. This context is used to give EQS a reference to the player character.

Figure 21. Blueprint logic of PlayerContext.

Most importantly, we then defined the environment query generator responsible for finding the desired cover points. The blueprint is shown in three different figures due to its size, Figure 22, Figure 23 and Figure 24.

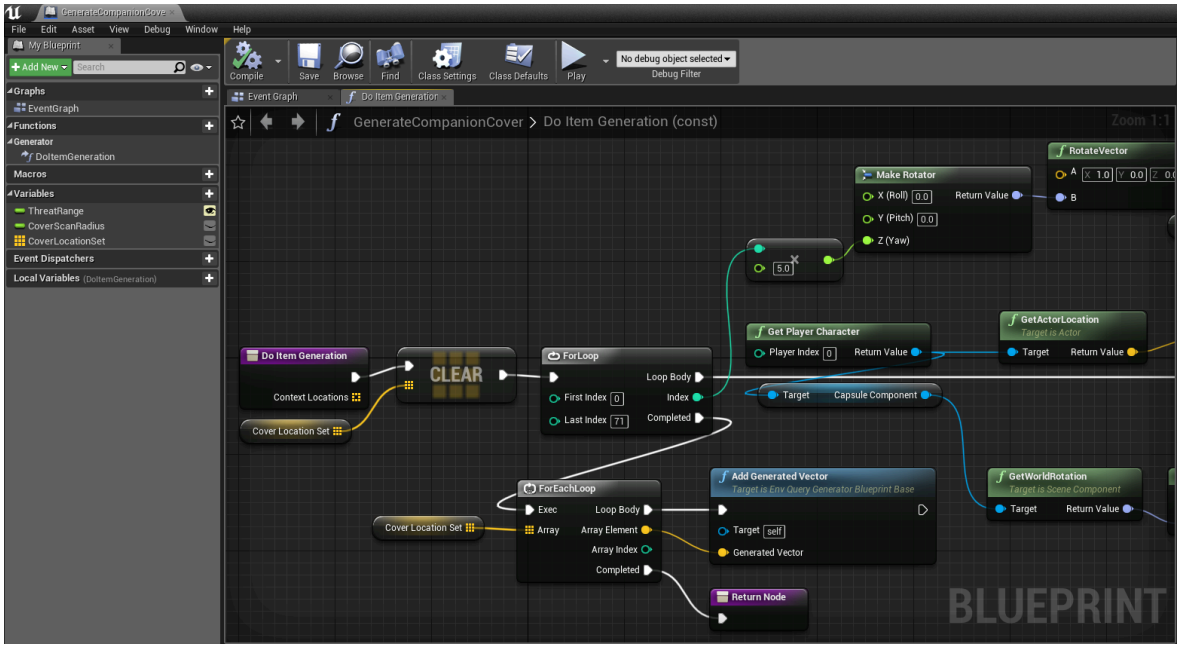


Figure 22. The leftmost third of the Blueprint logic of the GenerateCompanionCover query generator.

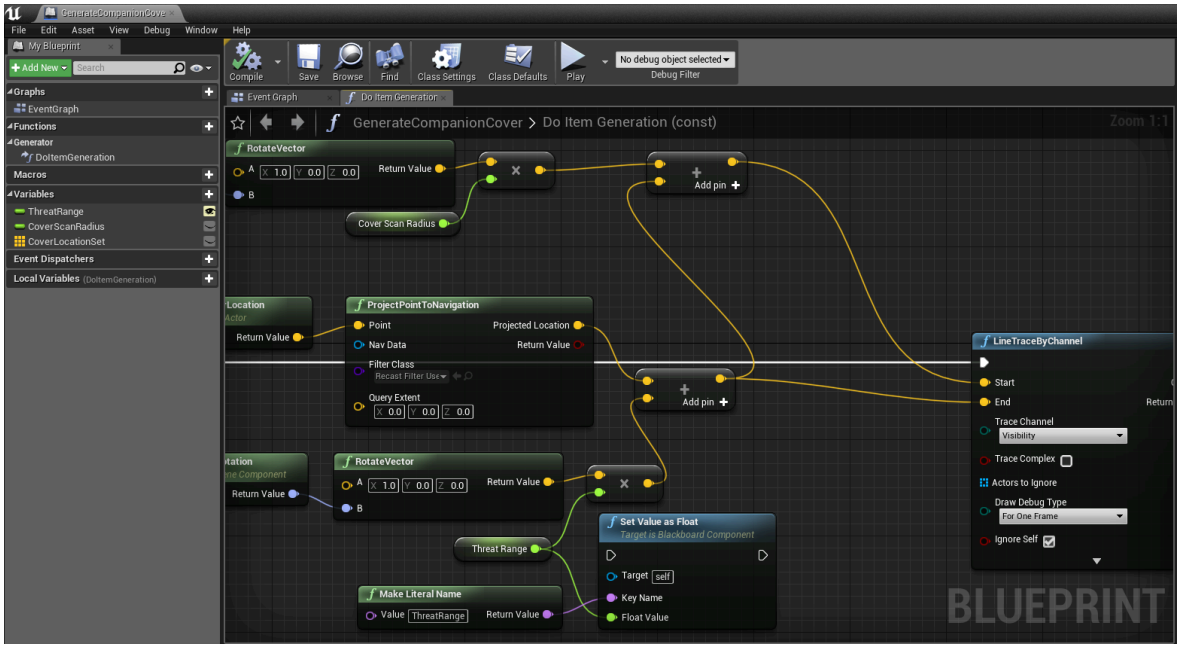


Figure 23. The central third of the Blueprint logic of the GenerateCompanionCover query generator.

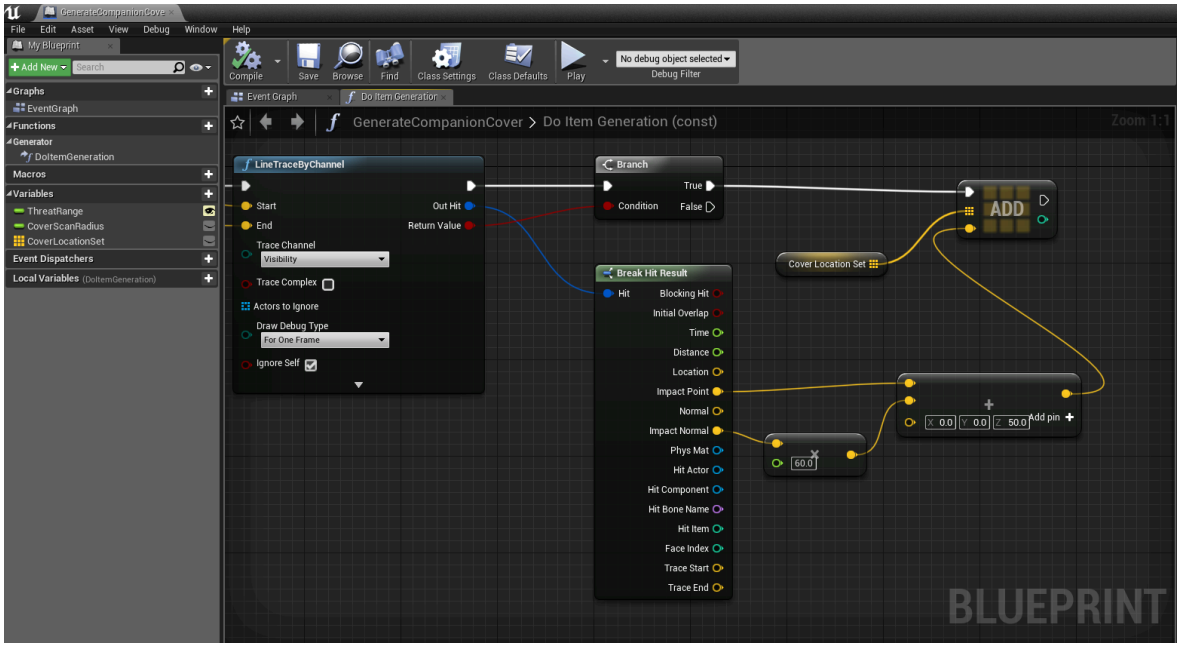


Figure 24. The last third of the Blueprint logic of the GenerateCompanionCover query generator.

The environment query generator is then used in the environment query itself, shown below in Figure 25:

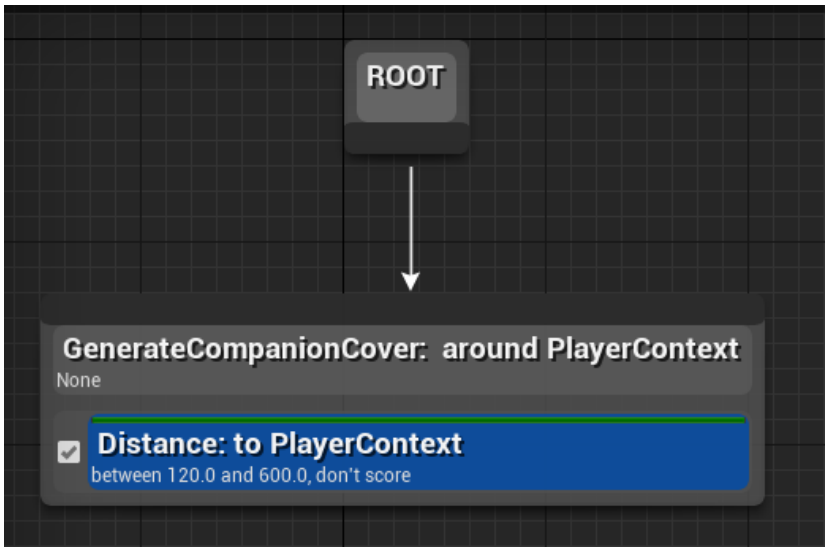


Figure 25. Blueprint logic of FindCompanionCover.

We specify certain settings not visible in Figure 25 as follows:

GenerateCompanionCover-node:

- Generated Item Type : Point
- Threat range : 600
- Cover scan radius : 800
- CoverLocationSet : Vector Array

Distance-test:

- Filter only
- Float min: 120
- Float max: 600

Lastly, the component responsible for executing the movement to the selected point is detailed below in Figure 26:

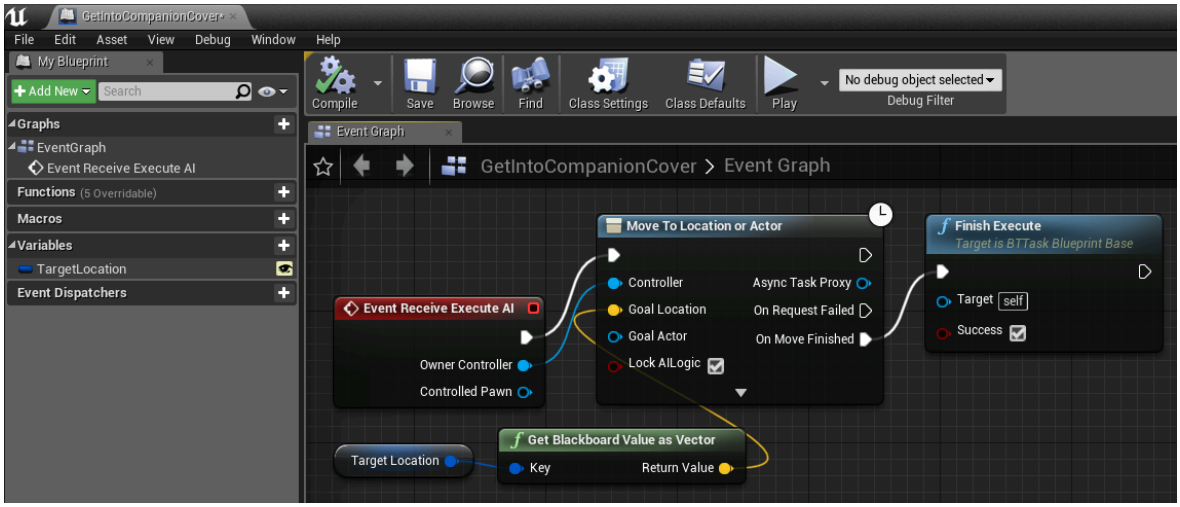


Figure 26. Blueprint logic of GetIntoCompanionCover.

The TargetLocation-variable is set as visible to editor, so that other components can access it.

4.2.2 SQS Complex Scenario

The same map was used as for the simple SQS scenario, and the settings from that scenario prevailed, such as the “Player”-tag on the player character.

The contents of the behaviour tree used by the AI agent are shown in Figure 27.

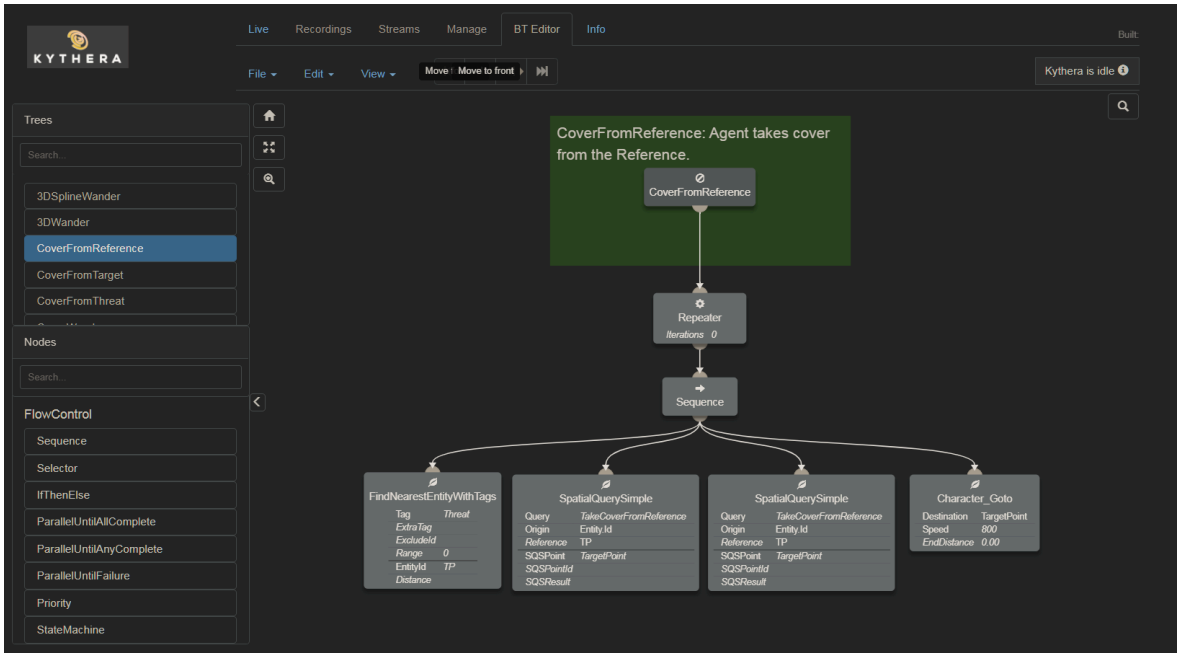


Figure 27. The CoverFromReference behaviour tree.

A KytheraTarget-object was placed as a child of the player character, about 800 units of distance away directly in front of the player. It was named “Threat”, given a Pawn-component named “ThreatPawn” and configured as shown in Figure 28.

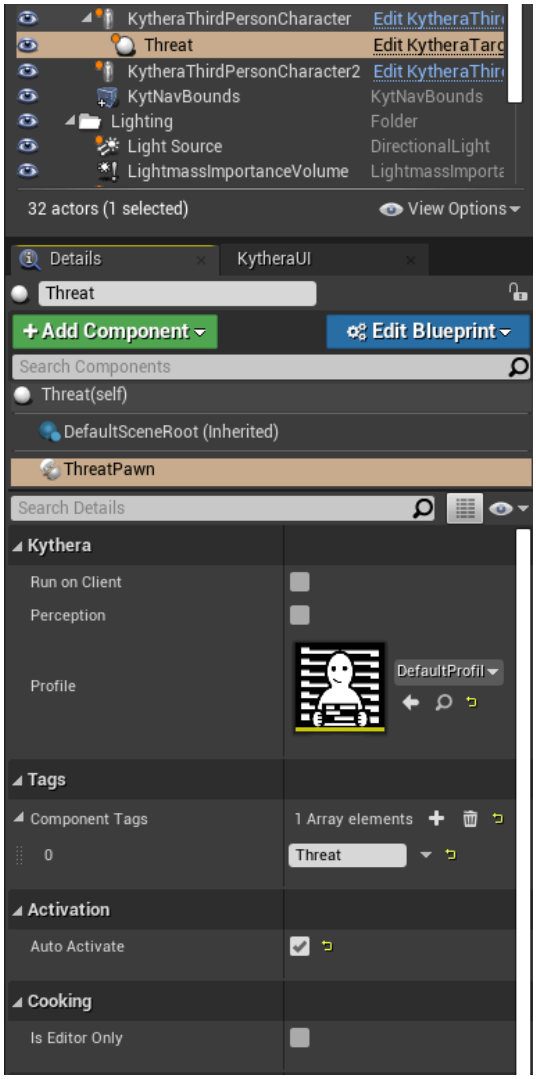


Figure 28. Details of the “Threat”-object’s “ThreatPawn”-component.

Code of the spatial query used is shown next. A generator generates points along cover rails around the reference object, the threat. Cover rails are created around obstacles that provide cover. Conditions are set so that the distance to the target, the player, is between two limits and that the acceptable points have no line of sight to the reference, or threat. Finally the points are weighted so that the ones closest to the player are preferred.

```
// Take cover while keeping close to the player.
SpatialQuery {
  Name("TakeCoverFromReference"),
```

```
  Generators{
    CoverRailGenerator(SpatialQueryObject::Reference,
                               "Full",1600.0f,400.f)
  },
  Conditions{
    LessThan(Distance3D(SpatialQueryObject::Target), 800.f),
    GreaterThan(Distance3D(SpatialQueryObject::Target), 200.f),
    NavRaycast(SpatialQueryObject::Reference, 0, 0.f, 0.f)
  },
  Scores{
    Weight(-1.0f, Distance3D(SpatialQueryObject::Target))
  }
}
```

A NavRaycast was erroneously used in the spatial query as it was an artifact left over from a previous query used as template. The error does not manifest in undesired agent behaviour however as all cover used in the scenario is full cover and there is no difference between a Raycast and NavRaycast. We suggest using a Raycast for this kind of scenario however.

4.3 Qualitative comparison and review of EQS and SQS and the development environments integrating them

Establishing and constructing two control scenarios allows us to observe what the use cases are like regarding the workflow and user experience in each scenario and what could be improved upon. The comparison and analysis in this thesis undoubtedly go hand in hand with a specific perspective, bias and background. The perspective and bias are that of a beginner UE4-, EQS- and SQS -user with little prior knowledge and experience of UE4 and no prior experience in EQS or SQS. The author's background is in IT, programming and hobbyist game development, but not in the use of these tools and development environments. The study itself does not contain a realistically complex scenario that could be found in today's released games, but touches the true potential of either system rather superficially. It would require an expert of both systems to deeply evaluate their strengths and weaknesses, and attempts to compensate for this have been made by citing the work of experts and asking for guidance from the field's pioneers. Furthermore, an important aspect such systems should strive to have is to be easily approached and understood by newer users and while powerful, should not be overwhelming to would-be developers. It could also be argued that within their feedback, expert users tend to ignore problems that they have learned to work around. Inexperienced eyes sometimes see potential for improvements in places where veteran users have internalized improvable aspects as a normal feature of the system.

In context of testing the EQS, while following the UE4 tutorials makes setting up the simple scenario rather easy it was noticed that power rarely comes without complexity. Certain blueprint blocks and selections were not very intuitive and a significantly larger amount of time would have gone into finding out the correct ways to build the blueprints without the tutorials. The blueprint tools, while powerful also present the user with large amounts of possible selections with little guidance. When creating blueprint logic and selecting the next blueprint part it was commonly one of three things that guided us to select a meaningful and correct one from the vast amount of selections the system offers. First, intuitiveness, or in other words the description and name of the piece would suggest it be selected. Second, the system itself suggesting what would fit via the context-sensitive menu. Third, the tutorial or guide associated with what was being designed. The more complex the design the less

often the two latter ones can be found available, thus it is important for the user themselves to be able to quickly assess the implications of each available selection from their name and description. Problems with making the correct selections are however assumed to be somewhat compounded by the inexperience of the author.

Use of the blueprint system alongside EQS seems intuitive, as it is well integrated into the environment and there are no extra considerations necessary when creating EQS queries. Some small steps took a moment to figure out due to ambiguities, but overall the official guide is well-detailed. Within thirty minutes we had a simple scenario running where one can play with the AI and observe its behaviour. The debugging tools and how the debug information is drawn on the screen are at first sight understandable and responsive, they are powerful in beginner hands as well. The connection between the environment query and associated debug spheres could clearly be seen and their implications understood. An example of this is when a green sphere among red spheres could be seen near the player, but not too close, on the same side of a separate adjacent cover wall as the player. This is clear indication of a successful result of an environment query that was designed to find the best location for the agent to move to while being covered from the direction the player is facing from behind their own cover, with an additional condition that it must not be too close to the player. The EQS debug coloring scheme of green for winning position, red for losing position and blue for a position culled by conditions is fine, although a color such as grey would be more intuitive instead of blue for a culled position.

Both development environments let the developer easily stack different tests, weights and conditions in their queries. For UE4's behaviour trees the developer can use tests pertaining to distance, dot product, gameplay tags, spatial overlap, pathfinding, projection or trace to more strctly and precisely select the winner for the query. For Kythera and its behaviour trees the options are more extensive and too numerous to list here. The selection of default behaviour tree nodes and query functions and parameters also suggests a stronger conceptual connection to what is usually used and required in FPS-games. Examples of this are nodes (and their associated tooltips) named "Character_IsPointReachableNow", which "succeeds if there is a valid path from the start point or entity to the end point or entity and fails if not", "Character_PredictPosition", which "predicts where an entity will be in x seconds assuming

they continue to move at constant velocity on the navmesh", and "HasCoverType", which "checks whether a cover point supports a particular cover type". The last one is a crucial property of cover in first-person shooters: cover could for example either be full, partial or soft cover.

Using criteria from Jack (June 2017), further defined in our discussions of said criteria, we can evaluate each system in a systematic way. Placing these definitions into a table format we can bestow a score of 1-3 for each criterion, accompanied by reasoning to justify the evaluation. While we attempt to be as objective and unbiased as possible to provide a result as useful as possible, it should be taken into consideration that the scoring is still based on the author's subjective evaluation and is meant to provide an easily digestible guideline of the system's ability to meet the criteria. A score of one is given if the attribute is found significantly lacking in relation to the defined criteria. A score of two is given if notable improvements to the attribute have been identified, but the criterion is still mostly met by modern standards and in comparison to the other system. A score of three is given if the criterion is mostly or fully met with no significant improvements identified. Any system used for choosing movement locations faces design pressures from many directions and must be flexible and expressive, as it will define the movement possible for the AI agents. It must be capable of rapid, iterative development, and it must include powerful tools as these will limit the quality of the final behaviours (Jack June 2017). In the next chapter, the EQS is analyzed and scored first, followed by SQS. A radar plot is also provided to condense the evaluation into one graphic.

5 Results

5.0.1 EQS Analysis

Flexibility - Tweaking and rearranging the logic and contents of the environment queries and the order of the tests within is easy, and their attributes are easily accessible in the graphical UI. Testing a change is a matter of clicking a drop-down menu or tweaking a value, saving the query, and launching the gameplay simulation. Changing the generator however requires deleting the query node. Tests can be dragged from a query to another, but only one by one. This is where a text-based editor is faster as you can easily copy and move text, but the same cannot be said for UI elements. When a custom generator or test is required, the developer needs to write C++ code and test it to add the generator to the list of generators. A slightly easier way is to generate the points via the blueprint system as we did for the EQS complex scenario which still requires significant effort. (”Unreal Engine: Custom EQS Generators” 2019). This slows down iterativity and makes extending, recombining and reapplying tests and generators harder, adding to the developer’s workload. A score of 1 is assigned.

Expressivity - EQS offers generators that generate points in various basic shapes such as a circle, cone, donut, grid or pathing grid as well as ones that generate actors of a certain class, or the current location of the querier. One can also make composite generators, that contain multiple generators inside. While these generators cover a variety of uses developers might have, they remain basic, are restricted to flat planes and lack the context-specific generators SQS offers. Additional generators need to be written by the developers to satisfy more complex needs which takes development time and slows down iterativity. A score of 2 is assigned.

Rapid iterativity - As mentioned above, EQS’ iterativity is affected by the work required to make new generators and tests. As a major part of the workload around creating a query comprises of the work on generators and tests, this is a major point for future improvement and highlights the necessity of a powerful query language. As a minor point, the variables, values and details of the query are behind drop-down menus and different panels of the graphical blueprint system. Being able to see everything at once, as is the case is with a text

editor would help the developer think and work faster. A score of 2 is assigned.

Efficiency - While the EQS' development environment gives the developer good control of the parameters defining the spatial extents of the query and the context it centers around, the small selection of default generators can easily lead to an unnecessarily large amount of generated points, each which needs to be tested and weighed. The location or entity around which the points are generated matters significantly in relation to efficiency. EQS seems to only accept Context-objects as these locations. While the queries are executed asynchronously, the filtering and scoring are executed synchronously, limiting efficiency. A score of 2 is assigned.

5.0.2 SQS Analysis

Flexibility - Rearranging and making changes to the spatial queries is a matter of rewriting, reorganizing, copying and pasting text. As the same file can contain multiple queries, being able to see examples of other similar functional queries while working on one is useful. The SQS documentation, unavailable for public citing at the time of writing this thesis, provides a vast array of generators, logical functions and operation functions to manipulate the parameters of the query with. Compared to those available in EQS, the difference is that of an order of magnitude. These options to manipulate the query contain both context-specific and spatially specific options in 2D and 3D space. The amount of options provided before any C++ needs to be written and the ease at which the query can be extended, recombined and quickly reapplied indicates that SQS is a good benchmark for flexibility. A score of 3 is assigned.

Expressivity - The large amount of options for a query's parameters compared to EQS and their context-specific nature allows for a developer to manifest and test new ideas quickly. The context-specificity of the language works in favor of at least first-person shooters, flight simulators, MMORPGs and perhaps even games of the strategy genre. Points can not only be created in various shapes, but also a cover rail system is in place. Cover rails are generated along the edges of obstacles to provide positions of cover. This is one of the ways in which SQS shows its context-specificity: in first-person shooters it is often a given that characters

position themselves next to obstacles to break line of sight from hostile characters and take cover from incoming fire. The cover rail system was noted as one of major timesavers during testing. A score of 3 is assigned.

Rapid iterativity - After approximately an hour of working on spatial queries and the simple scenario the workflow and time spent on cycles of tweaking the spatial query and behaviour tree was already reduced. This was further assisted by the powerful yet accessible debug-draw options of the Kythera AI Inspector view, where settings could be turned on to display useful information on practically any entity or variable that exists, and show the progression of behaviour trees, spatial queries and the results of the queries in real-time. Having all variables and values visible at once both in the text file and behaviour tree benefit the cognitive process and creativity of the developer. The previously mentioned amount of generators and testing options made it unnecessary to write any C++ code for the scenarios and working on the behaviour trees felt intuitive. In fact, no help, tutorial or consultation was needed for the final version of either the simple or complex scenario, a feat by the system that surprised the author. A score of 3 is assigned.

Efficiency - Asynchronous execution and context-specific features such as the cover rail system contribute to making generators in SQS efficient. As noted by Jack (June 2017), efficient queries go hand in hand with generators that generate points in specific locations that the developer is interested in. These reference points include the querier and any reference of entity or position provided to the query. Additionally, in comparison to EQS, which seems to only accept Contexts which must be pre-designed and assigned entities or locations as the generator's reference point, SQS uses a Target Selection system which extends the selection of valid references to those the AI agent perceives as its current target. Combined with behaviour trees one can search for any entity with a certain ID and feed it to the query, enabling low-effort and specific queries which ultimately results in high efficiency. A score of 3 is assigned.

Criterion	EQS Score	SQS Score
Flexibility	1	3
Expressivity	2	3
Rapid iterativity	2	3
Efficiency	2	3

Table 1. Scoretable for EQS and SQS

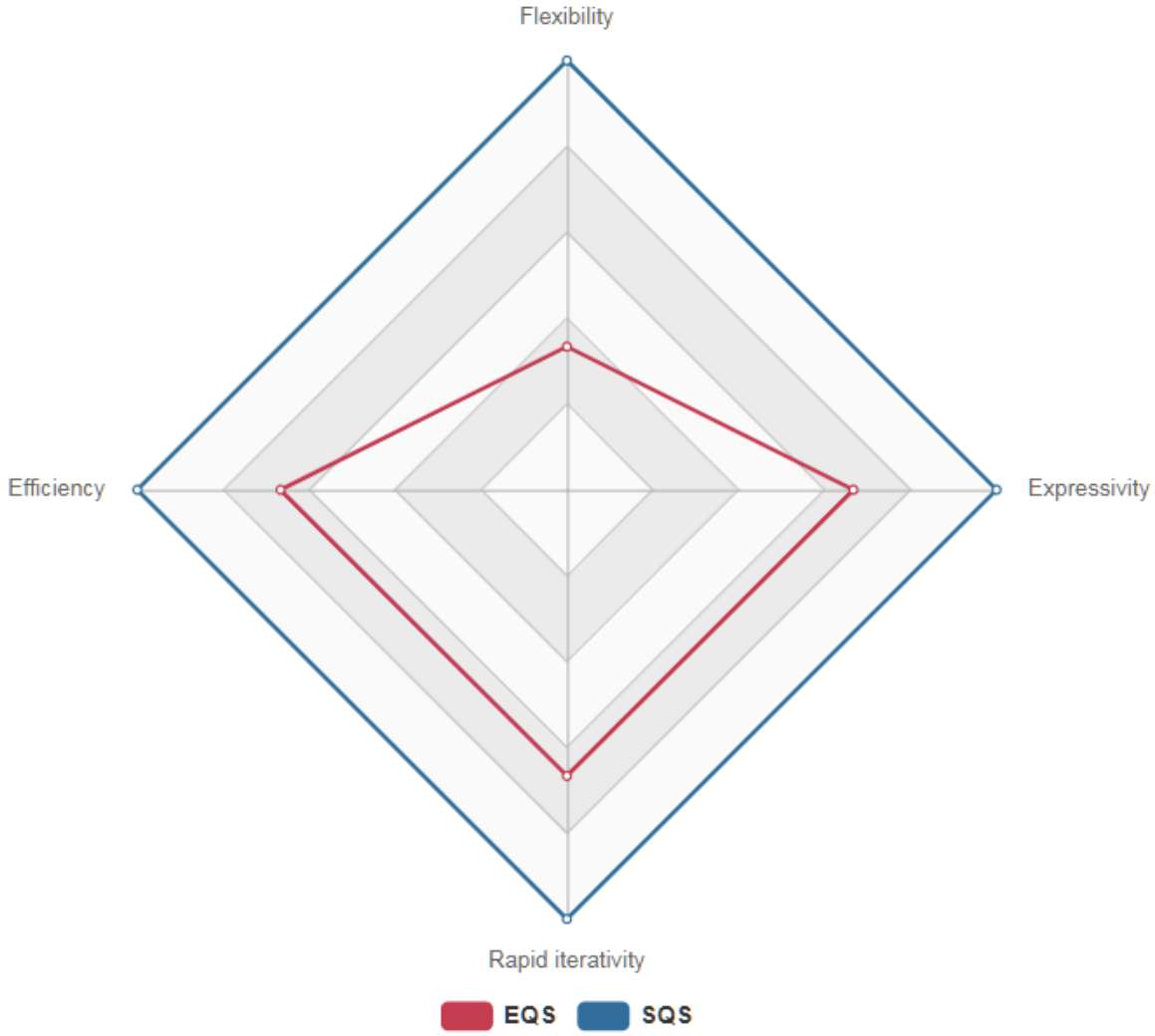


Figure 29. Radar plot of the evaluation scores.

6 Discussion

The results paint a noticeable difference between EQS and SQS. While the results are based on the evaluation by one person, the evaluation itself is based on expert definitions and criteria derived from literature on the topic. Because previous scientific literature and reviews were found to be scarce, we are not able to provide a convincing contrast between what this paper and previous research have found. Many of the articles in the Game AI Pro -book, the latest magnum opus of AI and AI design in games, deal with query systems and query languages. This gives us an indication of the direction that AI development is taking in the modern industry. Taking this into account, it is no surprise that SQS, being the more modern system, seems to tackle these modern problems more adequately according to the results.

The main improvement for EQS suggested by the results is to integrate a query language, so that the developer can to an extent avoid working on C++ code and instead write new queries in a query language. A query language is both easier to read for non-developers and humans in general, but it also contributes to quicker iterations and expressivity. The reduced effort required to implement changes and tweaks also encourages developers to be creative without the worry of having to see substantial effort for little change. The main improvement suggested for SQS is to increase the debugging information presented when processing a query file and to keep expanding the query language's "vocabulary".

We surmise that part of EQS' shortcomings stem from it being an experimental addition to the UE4 engine, while SQS has been built from the ground up as a full commercial feature. Regardless of the results, they relay useful signals of both what could be improved upon and what has been a success.

It is expected that people of different perspectives, expertise and experience might have a differing or opposite view on how the two systems rank by the defined criteria. The results produced here should at least serve as a stepping stone on the grander path of discussion about query systems, artificial intelligence and game development. We hope that developers of either system, or completely other systems found useful pointers, reasoning and sources by reading this thesis. We also hope that continued research into the topic encourages others

to provide additional comparison and discussion to accelerate the development of systems like these.

It is also necessary to consider any potential bias of this thesis towards Kythera AI. The evaluation criteria have been formed with assistance from Kythera AI and because of this it was thought paramount to be both transparent and open about the connection. However, we assert that the criteria are also universal and agreed on by experts in the AI field, most likely also by the UE4 developers. Virtues such as “flexibility”, “rapid iterativity”, “expressivity” and “efficiency”, as they are defined in this thesis, we think can be seen as important attributes in their own right regardless of reference to a particular firm, team or game engine.

Many aspects of this thesis could have been refined and extended with additional learning and expertise on the development environments and the two systems. However, we believe that the perspective of a new user and developer would have been diluted in the process and changed to a different, yet useful perspective. Also, what initially seemed a complex task was simpler, and the complex scenario could have been made even more complex to better demonstrate the power and capabilities of both systems. Initially the author was also intimidated by the lack of scientific papers found and it was considered that such a study might not have the proper academic legs to stand on. Fortunately, these worries were ballasted by the encouragement of industry professionals and pioneers indicating the need for such research and review.

For further research, a thesis dedicated solely to bring together and review the current literature and material on game AI environment queries and agency would be very useful for researchers of the field of artificial intelligence in games. Another branch of interesting research we suggest is the use of these systems and their extensions in other genres such as strategy games. Finally, industry professionals are encouraged to engage the scientific community to spark more interest for proper academic research.

7 Conclusion

The theoretical background and literature surrounding the query systems EQS and SQS were presented and the two systems compared with the help of criteria based on expert consultation. The Spatial Query System was found to be a more sophisticated and powerful system according to these criteria, while the Environment Query System could still serve developers with simpler needs adequately. The study also identified a need for more research in the topic and topics surrounding it.

Bibliography

”Amazon Lumberyard homepage”. 2020. Visited on November 28, 2020. https://aws.amazon.com/lumberyard/.

Dellermann, D., P. Ebel, M. Söllner, and J. M. Leimeister. 2019. ”Hybrid Intelligence”. *Business & Information Systems Engineering* 61 (5): 637–643. https://doi.org/10.1007/s12599-019-00595-2.

”Environment Query System: Overview”. No date. Visited on October 12, 2020. https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/EQSOverview/index.html.

Jack, Matthew. June 2017. *Tactical Position Selection, An Architecture and Query Language: Introduction.* Version 2017-06. Visited on October 3, 2020. http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter26_Tactical_Position_Selection.pdf.

Jack, Matthew, et al. August 2020. ”Spatial Query System: Overivew”. Visited on October 1, 2020. https://kythera.atlassian.net/wiki/spaces/KYTDOC/pages/254738436/Spatial+Query+System.

Johnson, Eric. June 2017. *Guide to Effective Auto-Generated Spatial Queries: Introduction.* Version 2017-06. Visited on November 29, 2020. http://www.gameaipro.com/GameAIPro3/GameAIPro3_Chapter26_Guide_to_Effective_Auto-Generated_Spatial_Queries.pdf.

Machado, Tiago, Daniel Gopstein, Andy Nealen, and Julian Togelius. 2019. ”Kwiri - What, When, Where and Who: Everything you ever wanted to know about your game but didn’t know how to ask”. 2nd Workshop on Knowledge Extraction from Games, KEG 2019 ; Conference date: 27-01-2019, *CEUR Workshop Proceedings* 2313:43–50. ISSN: 1613-0073.

Routio, P. 2007a. ”Comparative Study”. Visited on November 24, 2020. http://www2.uiah.fi/projects/metodi/172.htm.

———. 2007b. ”Modes of Knowing”. Visited on November 24, 2020. http://www2.uiah.fi/projects/metodi/148.htm.

Routio, P. 2007c. ”Normative Analysis and Preparing the Proposal”. Visited on November 24, 2020. http://www2.uiah.fi/projects/metodi/179.htm.

”Unreal Engine 4 EQS Quickstart Page”. 2020. Visited on November 27, 2020. https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/EQSQuickStart/index.html.

”Unreal Engine 4 EQS Start Page”. 2020. Visited on November 27, 2020. https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/index.html.

”Unreal Engine: Custom EQS Generators”. 2019. Visited on November 30, 2020. https://www.thinkandbuild.it/unreal-engine-custom-eqs-generators/.

Xia, B., X. Ye, and A. O. M. Abuassba. 2020. ”Recent Research on AI in Games”. In *2020 International Wireless Communications and Mobile Computing (IWCMC),* 505–510. https://doi.org/10.1109/IWCMC48107.2020.9148327.

Yannakakis, G. N., and J. Togelius. 2015. ”A Panorama of Artificial and Computational Intelligence in Games”. *IEEE Transactions on Computational Intelligence and AI in Games* 7 (4): 317–335. https://doi.org/10.1109/TCIAIG.2014.2339221.

Zielinski, Mieszko. June 2017. *Asking the Environment Smart Questions: Introduction.* Version 2017-06. Visited on October 1, 2020. http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter33_Asking_the_Environment_Smart_Questions.pdf.