Comprehensive Creative Technologies Project: A Dynamic Director Ai System for Shooter and Survival Games

Charlie Evans
charlie6.evans@live.uwe.ac.uk
Supervisor: James Huxtable

Department of Computer Science and Creative Technology
University of the West of England
Coldharbour Lane
Bristol BS16 1QY

Abstract

A Director Ai system built for use within shooter and survival games. The aim was to provide a generic system for designers to define their own rules and dictate the behaviour of the Director according to their own needs. The Rule Pattern was utilised to craft two simple rule engines, one for handling how the Director perceives the player's intensity and the other for handling it's behaviour. The project was successful in allowing a designer to craft rules for each genre, and providing a range of options for customising the director and active area set.

Keywords: director, ai, ai director, system, rule base, rules engine, rule pattern, rules

Brief biography

Interest for this project came from my love of the Left 4 Dead series and how the series never fails to entertain; with the game boasting so much replayability. After discovering it's Ai Director, I was curious as to whether such a system could be applied to other genres of games to produce interesting gameplay and good replay value.

Online portfolio: https://charlie2099.github.io/

How to access the project

Github repository: https://github.com/charlie2099/Director-Ai-For-Survival-and-Shooter-Games

The ReadMe located in the above repository details instructions on how to run the project.

Final video: https://youtu.be/Y1wvqfNqyqQ

1. Introduction

Maintaining player engagement and interest in a game is a goal all game designers endeavour to achieve and is one of the most fundamental aspects contributing towards the player's experience. Challenge is an important factor towards accomplishing this. "When there is nothing at stake, and if completing something is too hard, respondents do not want to keep playing" (Schoenau-fog, 2011, p.11). Challenge can induce the concept of flow, which can occur when "a person skills are fully involved in overcoming a challenge that is just about manageable" (Csikszentmihalyi, 1997, p.2). Thus, acting as a "magnet for learning new skills and increasing challenges" (Csikszentmihalyi, 1997, p.2). If a person is in a state of flow, then inevitably it suggests they must be fully engaged with an activity.

Turtle Rock's answer to this was with their 'Ai Director', a system that was popularised within their video game series Left 4 Dead, offering plenty of challenge and replay-ability to players (Turtle Rock, 2008). An Ai Director (or Director Ai) at its core is a collection of systems all working together seamlessly to provide a fresh and novel experience for the player, making each play session different from the last. It governs how the pacing and drama is carried out during the experience, and makes logical decisions of when, where, and how events should unfold in the game. It typically involves the director analysing the performance of a player, deciding when and where different enemies and items should be deployed based on a perceived stress or intensity level metric. The implementation of such a system proved to be successful, however it would be interesting to see how the system would perform in games of a different genre.

This project intends to create a generic director ai system for use within games of the action games; of which the shooter and survival game genres have been picked to conduct a feasibility study on due to their high popularity and fun game mechanics. These will be showcased in action within a top-down 2D style game for both. Unlike a specialised director ai system, this system will have the versatility to enable the system to be highly customisable regardless of if the game is a shooter or survival game. It will provide a system that will allow rules to be created by the designer to dictate how the director behaves; enabling designers to streamline the process of controlling its behaviour in an easy and versatile way.

The project deliverables are:

- Unity scenes for both the shooter and survival game
- A Unity package
- A progress log
- A project timeline

The project objectives are to:

- Develop a system that allows for rules to be created for both shooter and survival games
- Provide a versatile and highly customisable system that allows designers to sculpt the director according to their own needs

2. Literature review

The first piece of literature being reviewed was a public communication in the form of presentation slides presented by Booth (2009) during his 2009 Standford AIDE-og conference presentation. This source of literature provided a wealth of information, detailing the ai systems made for the videogame Left 4 Dead (2008) and how they contributed to the systems that make up the game's Ai Director; a system made for tailoring the player's experience to make it more interesting. Using this resource for research, tricky concepts could be much more easily

grasped as concepts were broken down with visual aids and bullet point lists noting the core ideas.

Two other influential pieces of literature to be reviewed are papers Applying Goal-Oriented Action Planning to Games and Three States and A Plan: The A.I. of F.E.A.R, which describe a method of handling character behaviour through a simplified variant of what is essentially a finite state machine. These papers bring to light the inner workings of the GOAP system as implemented within the videogame F.E.A.R (2005) and help to establish a grounding for improving the process of developing character behaviours through technical breakdowns and a discussion on context relevant case studies.

A further piece of literature was a blog by Ai and Games (Thompson, 2020) discussing how spreadsheets were employed in Watch Dogs 2 (2016) to create interesting and responsive behaviour from its civilian ai. This piece of literature was a useful resource as it provided a conversational-like overview of the different components and systems step by step whilst remaining technical. Diagrams and images were prevalent throughout which helped to visualise how the systems worked, which is great for learning new concepts than just reading pure text. Additionally, this resource comes from Ai and Game's Tommy Thompson, a reputable source when it comes to understanding the underlying background theory behind the workings of ai systems present in many types of games.

Another piece of literature to review is a blog titled 'Towards a Rule-Based Game Engine', which discusses ideas for the creation of a rule-based game engine that could accurately model rules and provide a workflow for designers to design games without the need for programmers (Pierce, 2011). Pierce unveils how games can be modelling and broken down into three elements: nouns, verbs, and rules. Nouns being elements such as variables, players, NPCs, and items, and verbs comprising the actions agents can perform such as a jump or an attack action. This blog discussion made a good resource for research, giving insight into how rules can potentially be modelled and processed by a system by generalising said rules down so that a generic system can be made.

A final piece of literature to review is 'An Efficient Rule-Based Distribution Reasoning Framework for Resource-bounded Systems', a journal article by Rakib and Uddin (2018). This was a particularly important paper that explored rule-based reasoning through a context aware systems development framework which includes a lightweight rule engine. Rakib et al. talk about how a rule-based system is structured, comprising a rule-base, inference engine, and working memory; with the rule-base containing a set of rules, working memory containing a set of facts and the inference engine controlling the system execution (Rakib, Uddin, 2018). The paper was beneficial in drawing an understanding of rule-based systems and their advantages, such as being stored separate from the code.

3. Research questions

The following research questions have been devised that will help drive the project forwards in the right direction:

• Can a system be written that is generic enough that rules for a shooter game and a survival game can be processed by a single system?

• How can a system be designed in a way that the process of creating rules is intuitive?

• Can a system be built to a standard that makes it applicable for use in a professional setting?

4. Research methods

To go about answering the research questions numerous research methods were deployed, consisting primarily of secondary research sources including journal articles, book chapters, conference proceedings, public communications, webpages and blogs. Journal articles and books were well utilised resources in developing a deeper and more in-depth understanding of the difficult concepts and systems required to create the director system. Papers were also selected that detailed existing projects and studies conducted that concerned rule-based systems and rule processing, in answering the first question.

Blogs were also a useful and well-used resource, providing a conversational-like and interesting discussion on concepts whilst remaining technical in nature. This enabled concepts to be grasped much more easily than reading straight from a book or textbook. Another much frequented resource were conference proceedings and public communications such as GDC talks, a series of technical, in-depth talks from reputable speakers in the game development industry. These were particularly helpful in gauging an understanding of technical concepts much like that of books and journal articles without the need for having to rely on reading to accumulate information.

Social networking websites and other social media related platforms were not an employed source of information due to the high potential for information inaccuracy and bias. Any webpages explored were carefully considered and fact checked against other sources to ensure the validity of the information given, and so was utilised to a lesser extent to other sources.

This secondary research was mostly qualitative research as knowledge had to be acquired on new concepts and theories that were not yet well understood. Quantitative research was not undertaken as formulating conclusions on any numerical data gathered through surveys, experiments or statistical data was unnecessary for the purposes of the project.

Primary research methods were not used in answering the proposed research questions due to the nature of the project, as there is a great deal of up-to-date existing research available which is easily accessible and relevant to the subject area and context. No new data was required to be collected as the system intended to be produced is built upon a system that already exists. The time that would be required to carry out interviewing, preparing questionaries and analysing and collating results found through primary research methods would be quite substantial and would only be detrimental to this project.

5. Ethical and professional principles

This project presents no ethical concerns as no primary research methods were conducted that would require participants to provide sensitive data of any kind. All data input required is by the director system from the designer, which is strictly for game related purposes. The project will not negatively impact any individual, group, or community and instead it is hoped that it will make a positive contribution to the field of game design and help developers streamline their development further.

6. Research findings

Booth discusses how an active area set, shown in figure 1, can be used to restrict and handle entity population and destruction within the confines of the navigation areas around the player as they move through the environment (Booth, 2009). Flow distance is also used as a metric described by Booth for populating enemies and loot in the environment based on the "travel distance from the starting room to each area in the navigation mesh" (Booth, 2009). Booth reveals how both tools contribute towards generating structured unpredictability, an important concept with the aim of promoting replayability. From this it can be said that the Director's interesting and seemingly random spawning of entities within the environment relies on smart methods of dictating spawning based on distance metrics.

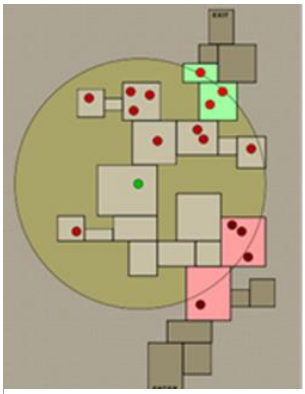

Fig 1: Left 4 Dead's Active Area Set

A potential approach for recreating an Active Area Set much like the one present within Left 4 Dead (2008) could involve use of the object pooling design pattern. This would deal with the re-use of assets when an entity goes outside the bounds of the active area. A metric for measuring the distance between the player and game world entities, as well as a predetermined bounds radius specified by the designer, can be used to set the size of the active area as well as dictating spawn locations of these entities within the world.

Orkin (2006) disclosed how Goal-Oriented Action Planning (GOAP) can be utilised to deliver real time behaviour planning for enemy agents so that more complex behaviour can emerge. GOAP is described as a "decision-making architecture" that enables agents to formulate plans in order to satisfy their own goals (Orkin, 2006). This approach to handling agent behaviour negates the need for pre-determined behaviour which can often be predictable and repetitive and decouples goals from actions making any new goals or actions easy to add. Following this approach will offer adaptable behaviour to best suit the agent's situation. It is important to note that the constant process of agents reformulating plans could present a slight performance overhead, especially when the project scales up. This could be mitigated to some extent through the deployment of the Active Area Set where entities will only exist within the confines of the active navigation areas surrounding the player.

A technique in which a director can craft an interesting experience is by creating interesting events and scenarios. In his blog discussing the systems that Watch Dogs 2 (2016) uses to make it's civilian ai behaviour, Thompson (2020) explains how attractor systems can be employed. Attractors are special activities placed in the world by the designers and are responsible for triggering a response from non-player characters; or the player themselves.

"Relying solely on the player to trigger in-world events is risky, given that the rest of the world is quite static and lifeless" (Thompson, 2020). "Provocative agents" are then used with the intent of provoking the attractor to trigger an event, which helps to bring more life to the world. From this research, an idea was formed for an implementation of such an attractor system. Attractors could be created by the designer with a Unity editor tool, or generated by the director itself, and then placed into the game world. These attractors could then react to any agents

designated as 'attractable', making use of c-sharp interfaces, and then trigger a change in the game that could either benefit or detriment the player.

From the findings of Pierce's (2013) blog on a rule-based engine, the proposed system could potentially feature a similar method for modelling rules; defining three elements, nouns, verbs, and rules that can be defined using Unity window editor tools. This design will help contribute towards the genericness of the intended system, enabling designers to add rules as they please. As well as this, tools could be designed as such that designers can pick and choose features of the director that they wish to include such as an active area set or a series of attractors.

Later in the development of the project research led to the discovery of the rule pattern. The rule pattern is a popular design pattern utilised by developers to help follow the principles of OCP, as well as the Single Responsibility Principle (SRP). By applying SRP, complicated rules can be abstracted from the rules processing logic and into their own rule classes. The rest of the system remains unchanged, applying OCP, as new rules are added.

Cyclomatic complexity is a quantitative measure of the complexity of a program, indicating the possible number of paths inside of a class or function. Typically, a cyclomatic complexity of 10 or below is acceptable in the case of methods. By using the rule pattern, the cyclomatic complexity of a method can be dramatically reduced.

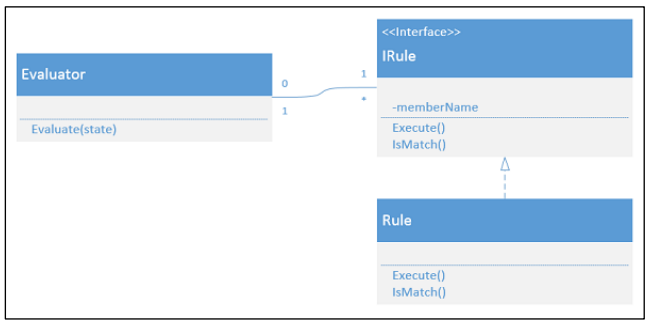

Fig 2: Diagram depicting the common rule pattern structure

The typical structure of a rule pattern consists of a rule interface, an evaluator, and the rule classes themselves (see Fig 2).

This discovery led to a more informed approached of how to implement the rule system for this project. Rules for this type of system can be abstracted into their own classes and make the process of creating rules much more intuitive for the designer.

7. Practice

7.1 Shooter and Survival Game Prototypes

Before creating the director, it was important that the shooter and survival prototypes were first setup to get a clearer vision of how the Director could be built to help make it generic in design.

7.1.1 Shooter Game

The first game prototype was a top-down 2D shooter in a tiled environment. The premise of the shooter game is to fight off hordes of enemies whilst attempting to activate all the generators.

7.1.2 Survival Game

The second and final game prototype was a top-down 2D survival game, also made in a tiled environment. The player needs to mine resources and gather items to craft and purchase tools to help survive the night. An inventory system was built as well as some simple mechanics for mining environment objects to obtain resources.

7.2 Director Design

The first step to building the director ai system involved designing the core director class that would manage all the systems that make up the director. It was important that this class supplied plentiful methods for grabbing state information for the various systems, as well as providing many options for customisability.

To monitor the performance of the player, the director needed to have some knowledge of, and therefore access to, player related data. Initially a generic template character was setup that included scripts for movement and camera handling. However, this started taking the project in another direction and into more of a pre-built toolkit rather than dynamic adaptive system. To rectify this the template character was scrapped and instead only a generic player script was made, 'PlayerTemplate',which was left to be modified by the designer as they wished. After the user attaches the script to their player character, the script could then be fed into the system through a serialised field in the director script's inspector view.

One potential drawback of this design however was that if a designer didn't want to use the template class and instead wanted to use their own class, then the designer would have to go and refactor any methods that had 'PlayerTemplate' as a parameter. This however was perceived to be of minimal concern as none of the methods make use of the argument by default as this is down to the designer when they configure their rules. Therefore, only a simple refactor would be necessary.

A central feature of the director is its intensity phase cycle. The director cycles through different phases of intensity which orchestrate how the drama unfolds. These states typically include a build-up, peak, declining peak, and a relax phase. The names chosen for this implementation were build-up, peak, peakfade, and respite.

These intensity states were defined in their own class for better code readability, providing getter methods for variables relating to the duration for each of the states. These were then serialised to allow easy modification for the designer.

Fig 3: Inspector view of the director script

Further options were provided to the designer through serialised fields in the inspector (see Fig 3). The designer can modify the default duration of the peak and respite phase as well as the default maximum population for each. The reason these are referred to as default is due to the potential for these values to be changed by the Director depending on the rules the designer may create for it. The threshold for when the intensity has peaked is also modifiable.

7.3 Active Area Set

One of the core systems that makes up part of the director is the Active Area Set (AAS). As previously covered, the AAS defines the active navigation area in which enemies, items, and events occur so that gameplay is contained only within the areas surrounding the player.

To help the designer easily identify and define the active area a line renderer component was attached to the director game object to produce a circular line using the following code (see Fig 4).

```
for (int i = 0; i < (segments + 1); i++)
{
    x = (Mathf.Sin(Mathf.Deg2Rad * angle) * radius) + playerPos.x;
    y = (Mathf.Cos(Mathf.Deg2Rad * angle) * radius) + playerPos.y;
    _line.SetPosition(i, position: new Vector3(x, y, z: 0));
    angle += (380f / segments);
}
```

Fig 4: Math for configuring the Active Area circle

By serialising the radius variable, the designer can specify how large they want the active area to be from within the inspector.

```
if (Vector2.Distance(a: _worldTilePositions[i], b: playerPos) < radius &&
    Vector2.Distance(a: _worldTilePositions[i], b: playerPos) >= cameraRadius)
{
    _activeTiles.Add(i);
}
```

Fig 5: Check for which tiles are designated as the active tiles

Working off a given tile map, the distance from each world tile to the player is checked to determine the active tiles surrounding the player (see Fig 5). If the tiles are within the radius of the active area set but also outside of the camera view radius, then the tile is added to a list of active tiles. Enemies are then spawned into a random tile from the active tiles list. Should any enemies stray outside of the active they are destroyed by the director and respawned to a closer location to the player.

As enemies will only exist within the active area set, it would be inefficient for a pathfinding navigation grid to be generated for the entire level. Making use of the Astar Pathfinding project by Aron Granberg, a navigation grid was implemented by assigning the provided 'Pathfinder' script to the director game object. The pathfinding grid is then set to the size of active area set at run-time. For some further optimisation the navigation grid is rescanned in intervals depending on the rate the designer has specified, rather than every frame which would induce major frame drops.

As well as enemies, the spawning of items should also be dictated by the director. The designer decides the spawn locations of items beforehand and should parent these items inside of an empty game object.

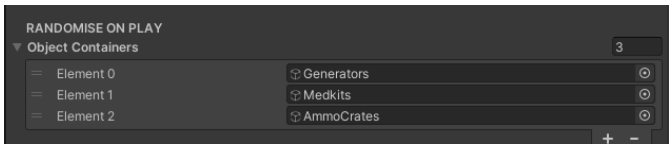

Fig 6: Item container list in the Director's inspector

A list of game objects was serialised for the designer to pass in a game object containing the items as child objects (see Fig 6). The director then iterates through each of the container's children and decides which items are active on each play by enabling, or disabling, the child object.

Now it was crucial that the intensity phase cycle was tested. A generic rule was created inside the director script that checked the distance of enemies from the player, based on the player object passed into the director by the designer. The perceived intensity value increases by a larger amount based on the closeness of an enemy to the player.

7.4 Debugger

To allow the designer to see information about the current state, intensity, and enemy population whilst debugging; a very simple ui debugger was made. These variables are exposed onto a user interface for the designer to see the workings of the director in real-time.

7.5 Rule System

7.5.1 Iteration After the First Prototype

After showcasing the first prototype of the project during the demo it was clear that some big changes were needed. Supervisors provided some very valuable feedback that led to the eventual discovery of a new approach for developing a rules system.

Beginning development on the new rule system a new concept was discovered that led to a simpler solution for implementing a rules system. This concept was the Open-Closed Principle (OCP). OCP is one of five design principles that make up the SOLID principles, an idea promoted by Robert C. Martin, that helps to make object-oriented designs more maintainable and understandable.

Meyer's definition of OCP states that "classes and methods should be open for extension but closed for modification" (Meyer, 1997). This means creating classes and methods whose behaviour can be changed without having to recompile and edit the code itself. This led to the discovery of the rule engine design pattern.

7.5.2 Intensity Rule Engine

The Director uses perceived intensity as an important metric for determining when to cycle through each state. Because of this, it was integral that the designer was able to define the rules that dictated how intensity was measured. First a rule interface was setup.

```
public interface IDirectorIntensityRule
{
    Frequently called  1 usage  6 implementations  Charlie
    public float CalculatePerceivedIntensity(Director director);
}
```

Fig 7: Interface class for the IDirectorIntensityRule

This interface acts a contract that says that all rules that derive from the interface must inherit its methods. The method for this interface returns a float value which will be the intensity output. The function also takes the Director as a parameter that will allow designers easy access to the director script's methods when creating rules.

Now that the rule interface was setup the first rule could be created, which for simplicity was a rule that checks if the player's health is below a value specified by the designer.

```
public float CalculatePerceivedIntensity(Director director)
{
    if (director.GetPlayer().GetCurrentHealth() <= _lowHealth)
    {
        return _intensity;
    }
    return 0;
}
```

Fig 8: Intensity rule for checking the player's health

As seen in figure 8, the player's health is accessed through the GetPlayer method that the Director possesses to have direct access to crucial player data. If the condition is satisfied the intensity value set by the designer is returned, otherwise 0 is returned meaning that there was no change in intensity.

For the rules to be processed a rule engine class was needed. The DirectorIntensityRuleEngine class was created for receiving a collection of rules and evaluating them to produce an output. The DirectorIntensityCalculator class was also created so that the designer could add and remove rules, deciding which rules would be included in the evaluation.

```
public DirectorIntensityCalculator()
{
    _rules = new List<IDirectorIntensityRule>
    {
        new HealthLowRule(10f, intensity: 6f),
    };
}
```

Fig 9: The constructor of the intensity calculator showing how rules can be added

Rules can be easily added by adding it to the list of intensity rules in the rule calculator's constructor (as shown in Fig 9). The designer can define constructor arguments which will allow rules to be easily tweaked later in development if necessary.

```
public DirectorIntensityRuleEngine(IEnumerable<IDirectorIntensityRule> rules)
{
    _rules.AddRange(rules);
}
```

Fig 10: Constructor of the intensity rule engine that takes in a rule collection

```
public float CalculatePerceivedIntensityPercentage(Director director)
{
    float intensity = 0;
    foreach (var rule in _rules)
    {
        intensity = Mathf.Max(a:intensity, b:rule.CalculatePerceivedIntensity(director));
        // Applies the rule which outputs the greatest intensity weighting
    }
    return intensity;
}
```

Fig 11: Intensity rule engine method for evaluating rules

Now that the rule had been added to the rule collection, the rule needed to be evaluated and processed to produce an output. Fig 11 shows how rules are evaluated according to whichever rule produces the highest intensity result. This means that when the director is measuring the intensity of the player it will increment, or decrement, the intensity value by the amount output by the rule system so that specific activities demonstrate to the Director more intense gameplay.

```
public float CalculatePerceivedIntensityOutput(Director director)
{
    var engine = new DirectorIntensityRuleEngine(_rules);
    return engine.CalculatePerceivedIntensityPercentage(director);
}
```

Fig 12: Intensity calculator method that uses the rule engine to output the evaluated intensity output

Back in the rules calculator class an instance of the rule engine was instantiated, passing in the rules collection through its constructor argument (see Fig 12). Finally, the rule engine's method was called to evaluate the given rules and provide an output. This output was used to calculate the perceived intensity (as seen in Fig 13) per a pre-set calculation rate that can be defined by the designer.

```
float intensity = _intensityCalculator.CalculatePerceivedIntensityOutput(director:this);
_perceivedIntensity += intensity * _intensityScaler * Time.deltaTime;
```

Fig 13: The intensity metric using the intensity calculator to determine its value

7.5.3 Behaviour Rule Engine

This rule system works well by itself for determining when the director should switch to a different intensity cycle, however at this point there was no way for the designer to control how the director behaved during these different phases. To work alongside the intensity rule engine a new rule engine was created, the director behaviour rule engine. This second rules system was setup in much the same way as the first, although differed in how the rules were evaluated by its rule engine.

```
public void CalculateBehaviourOutput(Director director)
{
    foreach (var rule in _rules)
    {
        rule.CalculateBehaviour(director);
        // Applies every rule
    }
}
```

Fig 14: Behaviour rule engine method for evaluating rules

Unlike the intensity rule engine which outputs the highest intensity value from any satisfied rules, the behaviour rule engine outputs ALL rules whose conditions are met (see Fig 14).

```
if(director.GetPlayer().GetCurrentHealth() >= director.GetPlayer().GetMaxHealth()
   && director.GetDirectorState().CurrentTempo == DirectorState.Tempo.Peak)
{
    director.maxPopulationCount += 4;
    director.SpawnBoss();
}
```

Fig 15: Rule that decides when a boss should be spawned

This allows for the possibility of several rules firing at once, that could pave the way for some potentially interesting behaviour to occur.

7.5.4 Improving How Rules Are Added

Although the current process of adding rules is straightforward, it requires the designer to touch the rule calculator scripts directly so that they can add in their rules. One solution to mitigate this is by utilising reflection.

```
var ruleType = typeof(IDirectorBehaviourRule);
IEnumerable<IDirectorBehaviourRule> rules = GetType().Assembly.GetTypes() // Ty
    .Where(p:Type => ruleType.IsAssignableFrom(p) && !p.IsInterface) // IEnumerabl
    .Select(r:Type => Activator.CreateInstance(r) as IDirectorBehaviourRule);
_rules.AddRange(rules);
```

Fig 16: Example of reflection being used to add rules automatically

As demonstrated in figure 16, reflection searches the codebase for classes that inherit from the behaviour rule interface but ignoring the interface itself. It then automatically creates an instance of each discovered rule. These rules are then added to the collection. With this code in place the designer no longer needs to be conscious of forgetting to add their rule as it will be automatically picked up by the system. The drawback to this technique however is that the rules should remain stateless. Although this can be worked around, it is much tricker to configure and is likely not worth the effort to do so. For this reason, the code was left commented out in case a use was found for it later in the development of the project.

Unlike the intensity rule engine that is calculated per the calculation rate defined by the designer, the behaviour rule engine is calculated on an

event basis. Every time the director switches to a new intensity phase state an event is invoked. This event is subscribed to and calls a method that contains the call to the behaviour calculator method.

7.5.5 Experimenting with Multiple Rules

With the rule system built it was now time to test the system fully by introducing more rules. Intensity rules for the shooter game were first created. These included rules such as checking if the player is idle, how low their health is, as well as fast they are obtaining kills (see Fig 17).

```
_rules = new List<IDirectorIntensityRule>
{
    new DistanceFromEnemyRule(3f, intensity: 6f),
    new DistanceFromEnemyRule(6f, intensity: 2f),
    new PlayerIdleRule(5f, intensity: 3f),
    new HealthLowRule(50f, intensity: 2f),
    new HealthLowRule(10f, intensity: 6f),
    new KillSpeedRule(killsTarget: 2, timeToAchieveKills: 6f, intensity: 5f)
};
```

Fig 17: Intensity rules for the shooter game

New rules for the director's behaviour within the shooter game were also devised. These included rules for spawning bosses and items, as well as how the player is progressing through objectives and if they are on a killstreak (see Fig 18).

```
_rules = new List<IDirectorBehaviourRule>
{
    new BossSpawningRule(),
    new MedkitSpawnOnPeakEnd(),
    new AmmoSpawnOnPeakEnd(),
    new KillStreakRule(killsToGet: 2, enemiesToSpawn: 5),
    new ProgressionRule(generatorsOnline: 2, enemiesToSpawn: 3)
};
```

Fig 18: Behaviour rules for the shooter game

```
_rules = new List<IDirectorIntensityRule>
{
    new DistanceFromEnemyRule(3f, intensity: 8f),
    new DistanceFromEnemyRule(6f, intensity: 4f),
    new PlayerIdleRule(5f, intensity: 3f),
    new HealthLowRule(50f, intensity: 2f),
    new PlayerAggressionRule(timePassed: 5f, intensity: 2f),
    new ResourcesSpentRule(resourcesGathered: 30, intensity: 4f),
    new ConsumableUseFrequencyRule(consumablesUsed: 2, timePassed: 5f,
};
```

Fig 19: Intensity rules for the survival game

New intensity rules for the survival game were also established (see Fig 19). Rules for this consisted of similar distance and health checking rules, along with the addition of unique rules such as how many resources the player has or how many consumables they have used.

The addition of these new rules began to shape the director into something much more interesting as playthroughs started to playout a little differently from the last.

8. Discussion of outcomes

The original aims for the project were to develop a highly customisable director ai that could be adapted to accommodate to a designer's needs by allowing them to create their own rules. This project has been successful in this regard. The project was able to demonstrate that rules manufactured by the designer for both shooter and survival games were able to be processed by a single system which achieved the primary goal of the project.

In the beginning stages of development however, the project saw major difficulties in creating a rule system that could be considered generic. After the submission of the prototype demo and receiving valuable tutor feedback the project was able to move forward with a clearer vision of where to go next. From there the project was able to see success as development on the rules system began.

Some of the key successes from the project include its easy rule creation system. Two rule engines were constructed providing the director as a context enabling rules to be defined with ease. The creation of rules is also aided by the methods defined within the designer's player class allowing rule classes easy access to player state information. The inclusion of the rules system meant that conditional logic related to the director's behaviour could be abstracted into their own rule scripts, which makes the codebase much more readable and easier to follow.

Another key success of the project was the accessibility and capability of the Active Area Set. Designers can pass in their own level tilemaps and the Active Area Set will use this to define all the possible spawn locations for enemies, removing the need for the designer to have to set each enemy's spawn position themselves. Enemy prefabs can be easily passed in and the Active Area Set will randomly select which enemies will spawn and will also spawn boss enemies based on the rules defined by the designer.

A limitation of the existing rules system is that it requires designers to manually add their rules to the rule calculator classes so that the rules are used in the rule engine calculations. Two potential solutions could be explored to fix this limitation. The first would be to make use of Reflection that would automatically pick up rule scripts created by the designer, create an instance of it, and then add it to the list of rules.

Another alternative would be to design a user interface that would interact with the rules system engines and allow rules to be added and removed through the interface. A benefit to this method is that it could also provide the option for enabling and disabling which rules are active. Something that could have been done better is introducing more rules to the survival game prototype to better test the director and induce more interesting behaviour. Unlike the shooter game there was a lack of variety in the created rules which.

A further way that the system could have been improved is by introducing more ways of deciding how rules are fired. In its current state rules are fired in one of two ways. Intensity rules are fired based on the highest intensity weighting output, and behaviour rules are all fired in parallel. Some alternative techniques of rule evaluation include firing the first rule that matches, executing a random rule, and firing rules in order: which can be applied by using LINQ statements. A benefit to using LINQ statements is its code readability and optimisation, demonstrated by Weimann who shows how code for finding the closest game object to the player can be shortened down to a single line of code (Weimann, 2017). However, some consideration should be taken for the small amount of garbage that they generate. Because of this they should be avoided from being called every frame in the update method (Weinmann, 2017).

A direction that the project would like to have explored was into designing custom editor tools. Although this was outside the scope of what was required for the project, custom tools could provide an alternative means of customising the director system whether that be through rule creation, or through populating the active area set and director scripts with data. This would benefit the developers by allowing them to streamline the development of their director system through a simple interface.

The design of the active area set system draws similarities to the one implemented within the Horror Ai Toolkit, a unity project built to help developers manage the creation and behaviour of their ai (Hickery, 2018). The active area set implementation seen in the Horror Ai Toolkit also makes use of a line renderer to visualise the active area, of which is customisable in in the unity inspector. A plethora of options are also present in the inspector of its director script for modifying the stress levels and behaviour of the ai. Where this project excels in comparison to the Horror Ai Toolkit however is in its deployment of a rules system to manage intensity and behaviour. The rules that affect how stress is calculated is hardcoded directly into the director script of the Horror Ai Toolkit. For example, it features a hard coded rule for increasing stress when the player is close to an enemy; a rule which is extracted into its own 'DistanceFromEnemy' rule class within this project. Furthermore, how the stress (or intensity) is measured is entirely up to the designer within this project's implementation which leaves more choice in the hands of the designer.

Unfortunately, the extent of the director system wasn't fully explored within the survival game as arguably too much focus and time was put into developing the shooter game. The survival game would have benefited from a more diverse set of rules; however, this does not disprove the fact that the system is capable of processing rules for this genre. So long as the designer provides methods for obtaining the relevant state data relating to the game, rules can quite easily be created for this genre. Where this project excels is in how the project could be adapted to other genres. A racing game for example may exhibit a rule for increasing the intensity when another racer is overtaking the player. This rule could quite easily be adopted with the existing system.

A further omission from the project that was intended to be employed was the attractor system that was detailed in the research findings. Further development on the project would explore realising this feature, with the existing system open to adaptation to this addition due the way the active area set is designed. The active area set could take in a container filled with attractor game objects, which have already been placed in the world by the designer, and the director will pick which ones are active on each play. The system would need to be adapted a little further to allow the attractors to interact with the director, perhaps providing attractors built-in as part of the director ai package. The attractors could then interact with the director through unity Action delegates which could call upon the behaviour rule engine to provoke an event to occur.

This project would be more applicable to a professional setting if more options were provided for debugging the director. In its current state it's limited to displaying the current state of the director on a simple ui overlay, however, to make it more accessible to developers', alternative solutions should be investigated. This could perhaps include an automated unit testing system that could automatically test rules that a designer creates to check that they perform as they should. This system could then output this information to the designer to let them know if their rules have passed the unit checks.

9. Conclusion and recommendations

The director system that has been produced has achieved success; delivering a simple but capable system that enables designers to create their own rules for the director in both shooter and survival games. Furthermore, the system provides a whole range of customisability that allows designers to tweak parameters and metrics as they see fit, tuning the director to their needs.

If the project were to be continued further, a complete user interface should be built that would allow a designer to add and remove rules with ease. This would negate the need for designers to have to enter the rule calculator scripts directly to filter which rules were active and provide a clean alterative to creating rules outside of the program.

Further work could also explore how the Active Area Set could be optimised to include more efficient techniques for checking whether enemies are in the bounds of the active area and how they could be pooled to boost performance. It would also be interesting to see if the Active Area Set could be upgraded to work with 3D games as in its current state the system is confined to the constraints of a 2D game.

Finally, it would be very interesting to see whether the director could be adapted to create mood and tension through visual effects or music. Much like how Left 4 Dead featured a lesser known second director that specifically dealt with audio and music management.

The project helps to demonstrate that a director system can in fact be implemented into more than just the shooter genre. It hopes to raise awareness that such a system is feasible for deployment with many other game genres, and that director systems can hopefully become more commonplace in the game development scene.

10. References

Figure 1 – Booth, Michael, 2009. The Ai Systems of Left 4 Dead. pp. 67. Available from: https://steamcdn-a.akamaihd.net/apps/valve/2009/ai_systems_of_l4d_mike_booth.pdf [Accessed 16 October 2021].

Booth, Michael. (2009). The Ai Systems of Left 4 Dead [presentation]. 17 November. Available from: https://steamcdna.akamaihd.net/apps/valve/2009/ai_systems_of_l4d_mike_booth.pdf [Accessed 16 October 2021].

Csikszentmihalyi, M. (1997) Finding flow: The psychology of engagement with everyday life. *Basic Books* [online]. p.2. [11 October 2021]

Hickery (2018) Horror Ai Toolkit. Available from: https://hickery.itch.io/unity-horror-ai-toolkit [Accessed 26 Nov 2021].

Meyer, B. (1997) *Object-oriented Software Construction*. 2nd Edition. United States: Prentice Hall.

Monolith Productions (2005) F.E.A.R. [Video game]. Vivendi Universal Games, Warner Bros. Games.

Orkin, J. (2003) Applying Goal-Oriented Action Planning to Games. *AI game programming wisdom* [online]. p. 3 [Accessed 27 October 2021].

Orkin, J. (2006) Three States and a Plan: *The Ai of F.E.A.R. Game developers conference* [online]. p. 6 [Accessed 27 October 2021].

Pierce, Shay (2013) *Towards a Rule-Based Game Engine*. Available from: https://www.gamedeveloper.com/design/towards-a-rule-based-game-engine [Accessed 16 November 2021].

Rakib, A., Uddin, I (2018) An Efficient Rule-Based Distributed Reasoning Framework for Resource-bounded Systems. *Mobile Networks and Applications* [online]. [Accessed 20 October 2021]

Schoenau-fog, H.S. (2011) The Player Engagement Process – an Exploration of Continuation Desire in Digital Games. *Digra Journal* [online]. 6, p. 11. [Accessed 11 October 2021].

Thompson, T. (2020) How Spreadsheets Power Civilian AI in Watch Dogs 2. Case Studies [blog]. 5 November. Available from: https://www.aiandgames.com/2020/11/05/howspreadsheets-power-civilian-ai-in-watch-dogs-2/ [Accessed 12 November 2021].

Turtle Rock Studios (2008) Left 4 Dead. [Video game]. Valve Corporation.

Ubisoft Montreal (2016) Watch Dogs 2. [Video game]. Ubisoft.

Figure 6 – Whelan, Michael. (2013). Behaviour Driven Blog. *The Rules Design Pattern* [blog]. 14 May. Available from: https://www.michael-whelan.net/rules-design-pattern/ [Accessed 19 March 2022].

Weimann, Jason. (2017). Unity3D. *LINQ for Unity Developers* [blog]. 1 July. Available from:

https://unity3d.college/2017/07/01/linq-unity-developers/ [Accessed 04 April 2022].

11. Bibliography

Ai and Games (2020). Building the AI of F.E.A.R. with Goal Oriented Action Planning | AI 101. YouTube [video]. 06 May. Available from: https://youtu.be/PaOLBOuyswI [Accessed 5 November 2021].

Ai and Games (2019). Behaviour Trees: The Cornerstone of Modern Game AI | AI 101. YouTube [video]. 02 January. Available from: https://youtu.be/6VBCXvfNlCM [Accessed 10 December 2021].

Booth, Michael. (2009). Replayable Cooperative Game Design: Left 4 Dead [presentation]. 17 November. Available from: https://cdn.cloudflare.steamstatic.com/apps/valve/2009/GDC2009_ReplayableCooperativeGameDesign_Left4Dead.pdf [Accessed 16 October 2021].

GDC (2017). Goal-Oriented Action Planning: Ten Years of AI Programming. YouTube [video]. 09 October. Available from: https://youtu.be/gm7K68663rA [Accessed 5 November 2021].

Ishida, T., Stolfo, S. (1984) Towards the Parallel Execution of Rules in Production System

Thompson, T. (2014) In the Director's Chair: The AI of Left 4 Dead. Available from: https://medium.com/@t2thompson/in-the-directors-chair-the-ai-of-left-4-dead78f0d4fbf86a [Accessed 16 October 2021].

Thompson, T. (2020) Revisiting the AI of Alien: Isolation. Case Studies [blog]. 20 May. Available from: https://www.aiandgames.com/2020/05/20/revisiting-alien-isolation/ [Accessed 26 October 2021].

Programs. [online]. p. 3 [Accessed 2 December 2021].

Jacopin, E. (2014) Game Ai Planning Analytics: The Case of Three First-Person Shooters. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* [online]. p. 5 [Accessed 2 November 2021].

M.Bourg, D., Seemann, G. (2004) Rule Based AI. In: M.Bourg, D., Seemann, G. (2004) *AI for Game Developers* [online]. Sebastopol: O'Reilly Media, Inc, Chapter 11. [Accessed 26 November 2021].

Nystrom, B. (2011) Observer. In: Nystrom, B. (2011) Game Programming Patterns [online]. Genever Benning, 2014. [Accessed 7 December 2021].

Owens, Brent (2014) Goal Oriented Action Planning for a Smarter AI. Available from: https://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-asmarter-ai--cms-20793 [Accessed 5 November 2021].

Snowden, J., Oikonomou, A. (2011) Creating More Entertaining and Re-Playable Games By Dynamically Introducing and Manipulating, Static Gameplay Elements And Events. *2011 16th International Conference on Computer Games* (CGAMES) [online]. p. 3 [Accessed 12 December 2021]

Thompson, T. (2021) How AI Help Achieve Tactical Clarity in Gears Tactics. Case Studies [blog]. 29 April. Available from: https://www.aiandgames.com/2021/04/29/how-ai-helpachieve-tactical-clarity-in-gears-tactics/ [Accessed 21 November 2021].

Van Fleet, Ryan. (2019). Basic Rules Engine Design Pattern. [blog]. 12 November. Available from: https://tenmilesquare.com/resources/software-development/basic-rules-engine-design-pattern/ [Accessed 18 March 2022].

Appendix A: Project Log

Charlie Evans 18009251	**A Dynamic Director Ai System for Shooter and Survival Games**		
Date (w/o)	**Task**	**Outcome**	**Notes**
01/11/21	• Final touches to the project proposal	Project proposal submitted	
15/11/21	• Setup GitHub repository with directories for both Unity game projects	Two unity projects setup and pushed to a single GitHub repository.	

22/11/21	• Begin development on the shooter game • Design the level environment	A tilemap was used to create the level environment. A large level was made with a variety of wide and narrow areas.	
29/11/21	• Play around with events using Unity's Action event delegates	Unity's built in Action delegates were played around with and used to publish events that other scripts could subscribe to.	Could make use of Action events for handling the director intensity phases.
06/12/21	• Complete the first draft of the research document	Sent draft to supervisor Received some useful feedback	
13/12/21	• Finish up research document ready for submission	Research document submitted	
20/12/21	• Setup the core director scripts • Setup intensity phase loop	A director script was made providing several serialised inspector options. Basic intensity loop was setup which dictates when to switch to the next intensity phase	
27/12/21	• Implement the first prototype of the Active Area Set system	Active Area Set script was made and attached to the Director script alongside a line renderer component. The area circle colour and dimensions were made customisable in the inspector.	Should provide the option to pass in a list of enemies
03/01/22	• Generate a pathfinding grid the size of the active area set that the level environment scans periodically	A pathfinding script was added to the director. The pathfinding navigation grid was generated to the size and position of the active area set bounds and scans the level environment periodically to generate the walkable (and spawnable) areas.	Spawned enemies can sometimes spawn inside of walls. Enemies could be spawned onto tile positions of a specific layer.
10/01/22	• Setup a ui interface for displaying the current state of the director	A quick and simple ui interface was made that exposes a range of variables from the director such as the perceived intensity and intensity phase.	
17/01/22	• Finish prototype showcase video	Submitted the showcase vid. The demo showed off the Active Area Set system and customisability options.	
24/01/22	• Prepare for prototype demo	Prototype demo submitted	
07/02/22	• Begin development on the survival game • Mock up the level environment	A level environment was setup using a tilemap. Basic player movement and controls scripts made.	Inventory system?
14/02/22	• Build simple inventory system for survival game	A custom inventory system was designed. Resources can be picked up and are added to a inventory system of a fixed number of item stacks, of which each stack can contain items of one itemtype at a time.	
21/02/22	• Continue with development of survival game	Further development of the inventory system and integrating the active area set into the survival game.	
14/03/22	• Start on new rules system based on prototype demo feedback	Discovered the rule engine design pattern through the course of development on new rule system. Allows messy and hard-to-read conditional logic to be abstracted into their own rule scripts.	
21/03/22	• Implement a system based on the rule engine pattern	A rule engine was setup to help calculate the perceived intensity of the player.	Another rule engine could be setup that determines the behaviour or state of the director
28/03/22	• Design some rules for the shooter game to test the director	A rule for checking the distance from the enemy to the player was created.	
04/04/22	• Design some rules for the survival game to test the director	Rules for checking the player's health and hunger was added. A rule was also added for checking how full the player's inventory is.	
11/04/22	• Refactor and make code more readable	General code-up, blocks of code refactored into their own methods. Small code optimisations.	
18/04/22	• Reconfigure director intensity states into its own class	Director intensity phases refactored into their separate class. An instance of it was instantiated within the Director script.	An event could be invoked when there is a change in the intensity phase state.
25/04/22	• Add the option for the Active Area Set to take in a tilemap	A tilemap can be passed in to the AAS via the inspector. This layer determines the spawnable tiles for any spawned enemies.	
30/05/22	• Continue write-up of final report	More progress made on practice section of the report.	
06/06/22	• Re-write sections of the research findings	Research Findings sections updated with rules pattern discovery.	

	based on new findings		
13/06/22	• Continue with final report	Further progress made on practice section, updated earlier sections.	
20/06/22	• Integrate the latest director system into the survival game	Director and rule engine scripts updated to the changes that were made to the director within the shooter game.	
04/07/22	• Refactor all Director sub-systems into its own directory	Refactored director scripts to be under its own namespace. All scripts and files related to the director system was moved to their own directory.	Export as a package
11/07/22	• Collate the Director project into its own unity package	The Ai director was exported as its own unity package. A scene was included with an example of how to set it up. The package was uploaded to GitHub and provided in the project release.	
18/07/22	• Final touches before final project submission (resit) • Update Github repository readme	Updated github repository with information on how to setup and run the project, along with general information about the project.	

Appendix B: Project Timeline

Charlie Evans 18009251	**A Dynamic Director Ai System for Shooter and Survival Games**
Month	**Timeline**
November	Built a simple top-down 2D shooter game Playing around with event systems
December	Director script setup and designed Begun development on the Active Area Set
January	Continued with the development of the Active Area Set Project Prototype Demo
February	Built a simple top-down 2D survival game
March	Started work on a new rules system based on tutor feedback from the demo
April	Continuing work on the rules system
May	Not much work was conducted on the project during this month
June	Finalising prototypes before project submission Collating the Director into its own unity package
July	Resit submission

Appendix C: Assets used in the Project

Assets, Code and Software **NOT** produced by me:

- Unity Engine version 2021.1.21f, https://unity3d.com/get-unity/download/archive
- A* Pathfinding Project by Aron Granberg, https://arongranberg.com/astar/download
  - Script: 'Pathfinder' (attached to the Director script)
- Kenney Assets Top-down shooter pack, https://kenney.nl/assets/topdown-shooter

Assets Used In The Shooter Game:

- Kenney Assets Crosshair pack, https://kenney.nl/assets/crosshair-pack
- Zombie ui pack, https://opengameart.org/content/zombie-ui-pack
- 2D Sci-fi Platform Builder, https://f0x0ne.itch.io/2d-sci-fi-platform-builder

Assets Used In The Survival Game:

- Kenney Assets Pirate Pack, https://kenney.nl/assets/pirate-pack
- Palm Tree asset, https://opengameart.org/content/palmjungle-trees-for-32x32-tileset
- Cursor asset, https://kenney.nl/assets/puzzle-pack
- Treasure chest asset
- Food icons asset pack, https://opengameart.org/content/food-icons
- RPG Crafting Material Icons, https://opengameart.org/content/rpg-crafting-material-icons
- Rocks asset pack, https://www.clipartmax.com/download/m2H7Z5H7N4d3H7K9_rock-sprite-png-2d-rock-sprite/
- Voxel pack, https://kenney.nl/assets/voxel-pack
- Univseral Render Pipeline (URP), Built into Unity Editor, Unity Companion License

- TextMeshPro, Built into Unity Editor, Unity Companion License