โครงร่างวิทยานิพนธ์

(THESIS PROPOSAL)

ชื่อเรื่อง (ภาษาไทย) การใช้สถานะของเกมใน

ชื่อเรื่อง (ภาษาอังกฤษ) Implement Game State for Multi-Paths Quest in Structural Analysis Quest Generation

เสนอโดย นายธงธรรม จงมีสุข

เลขประจำตัว 5970176521

หลักสูตร วิศวกรรมศาสตร์มหาบัณฑิต

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะ วิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

สถานที่ติดต่อ 189 โชคชัย 4 ซอย 9 วังทองหลาง วังทองหลาง

กรุงเทพฯ 10310

โทรศัพท์ 084 112 1819

อีเมล์ Thongtham.th.Chongmesuk@gmail.com

thongtham.c@student.chula.ac.th

อาจารย์ที่ปรึกษา ผศ.ดร.วิษณุ โคตรจรัส

คำสำคัญ (ภาษาไทย) ,,,,,เกมที่ใช้จริงในเชิงพาณิชย์

คำสำคัญ (ภาษาอังกฤษ) PROCEDURAL GENERATION, QUEST GENERATION, COMPUTER GAME QUEST, GAME STATE, COMMERCIAL GAME

โครงร่างวิทยานิพนธ์

**หัวข้อวิทยานิพนธ์**

ภาษาไทย

ภาษาอังกฤษ Implement Game State for Multi-Paths Quest in Structural Analysis Quest Generation

**1. ที่มาและความสำคัญของปัญหา**

INTRO: EXPLAIN GAME AND QUEST AND PROBLEM

Quest is one of the essential part of Role-playing Game (RPG game) genre. Quest will inform player about what have to be done for the quest/story to go forward. Quest also record actions the player had already done which bring the player to the current state of the quest/story. In a way, quest log is similar to the player’s own journal on the game world and how the player had interact with it so far.

However, many computer RPG games’ quest are too restrictive on what the player can do to move the quest/story forward. Many quests present player with limited path of actions player could do to complete the quest. This restriction in player freedom of action could lower player’s immersion when they realize that other players must also perform similar action to complete the quest, thus rendering the ‘my unique journal’ of player to be not so ‘unique’ compare to each other’s.

Some quests try to minimize this restriction by informing the player of what has to be done (goal state) without specific what action have to be taken, allowing player to discover the way to achieve the goal by themselves. However, in many case there are only 1 action or a set of actions that could lead to that specific state of the game. It could result in the same effect when the player discover that fact.

INTRO: EXPLAIN how PCG is used to deal with the problem

In order to create more variety and unique experience for each individual player, procedural quest generation (PQG) is developed and implement into the game. PQG is a subset of procedural content generation (PCG), a system which create content within game automatically and randomly (non-determined). The main purpose of PCG is to present each player with unique content that randomly change every time the game is played.

Still, current commercial PQG (in the game such as Elder Scroll Skyrim, Fallout 4) generated quest by combining multiple pre-scripted part of quest together to create a new quest. This, while guarantee the generated quest to always be functioning and consistent, still limit the flexibility of player action and quest content. Therefore new way to generate quest for computer game are being researched and developed. One such work is the structural analysis by Doran and Parberry [2011]. Doran and Parberry analysis quest from 4 MMORPG games and conclude common structure of quest which can be used in quest generating.

The structural analysis is a quest structure rule which can be used to construct quest.

INTRO: Previous work

In the work of Doran and Parberry [2011], their prototype quest generation use Prolog language to create all possible path to complete the input quest using the analysed structure / ‘grammar’. Then a path is selected as the ‘generated quest’ and present it to the player. Their quest structure / ‘grammar’ is further expand in Machado, Santos and Dias’ [2017] work.

Other works in quest generation that use different approaches include…..……….

Lee and Cho’s [2012] Dynamic Quest Plot Generation using Petri Net Planning.

Jens van de Water’s [2011]’s A Framework for Formalizing Dynamic Quests.

Most researches aim to create unique quests on the game environment and world that are pre-determined.

This may allow the system to generate unique quest for each player, but it cannot guarantee freedom of action of the player on how to complete the quest. ………………………..

INTRO: EXPLAIN how this thesis can improve it and solve the problem better

This thesis propose to implement game state checking and action resolve into the quest generation which use structural analysis. This system will replace the part of quest generation that generate list of task/action (path) player character has to perform to reach game state where the quest condition is complete. The new replacement system will allow the quest generation system to determine exactly how many paths the player can actually take to reach the same quest complete condition. The new ability to measure player freedom of action should allow the quest generation system to generate quest with higher flexibility without compromising the integrity of the generated quest.

The new system developed in this proposal aim to

ระบบการจัดการเนื้อเรื่องที่ได้พัฒนาขึ้นสำหรับวิทยานิพนธ์ฉบับนี้ มุ่งหวังให้เกิดองค์ความรู้ที่สามารถเผยแพร่ใช้กับเกมที่มีขายตามท้องตลาดได้จริง

The path consist can be summarized into “game state”(node) and “action”(path). Each action will take game state as input, and deliver modified game state as output. …………….

**2. ทฤษฎีที่เกี่ยวข้อง**

**2. Related Theories**

Related Theories in quest generation are consisted of [Role-playing Games], [] , [].

ทฤษฎีที่สำคัญในงานวิจัยนี้ประกอบด้วย รายละเอียดของเกมประเภทสวมบทบาท การจำแนกประเภทผู้เล่น และทฤษฎีการสร้างเนื้อเรื่อง

**2.1 Role-playing Games and Quest**

Role-playing Games (RPGs) are a genre in game that originated from pen and paper board game. In RPGs, player will control character (one or multiple) to explore the game world and interact with it. The main goal of RPGs is not to ‘win’ the game, but rather ‘interact’ with the game world and observe how the game world will react to player action.

The term RPGs can also be used to describe game which has a progressive development mechanic in player’s character ability and equipment; such as unlocking ability to fly, or upgrading weapon to perform higher damage.

RPGs main focus element is narrative, exploration, strategic planing, and deep character interaction, rather than combat or precision timing. However, other genre may implement RPGs element to create sub-genre. For example, Action-RPGs such as Dark Soul series which emphasis more on real-time combat, and Stretegy-RPGs such as Final Fantasy Tactics series and Crusader King series which emphasis on complex planing and play closer to chess.

The narrative of RPGs story can be dynamic or static base on game story design. However, most RPGs story will have a main storyline which the whole narrative revolve around. This could be ‘saving your kingdom from Alien invasion’, ‘seeking your missing parent’, or ‘revealing the mystery of a certain anomaly’.

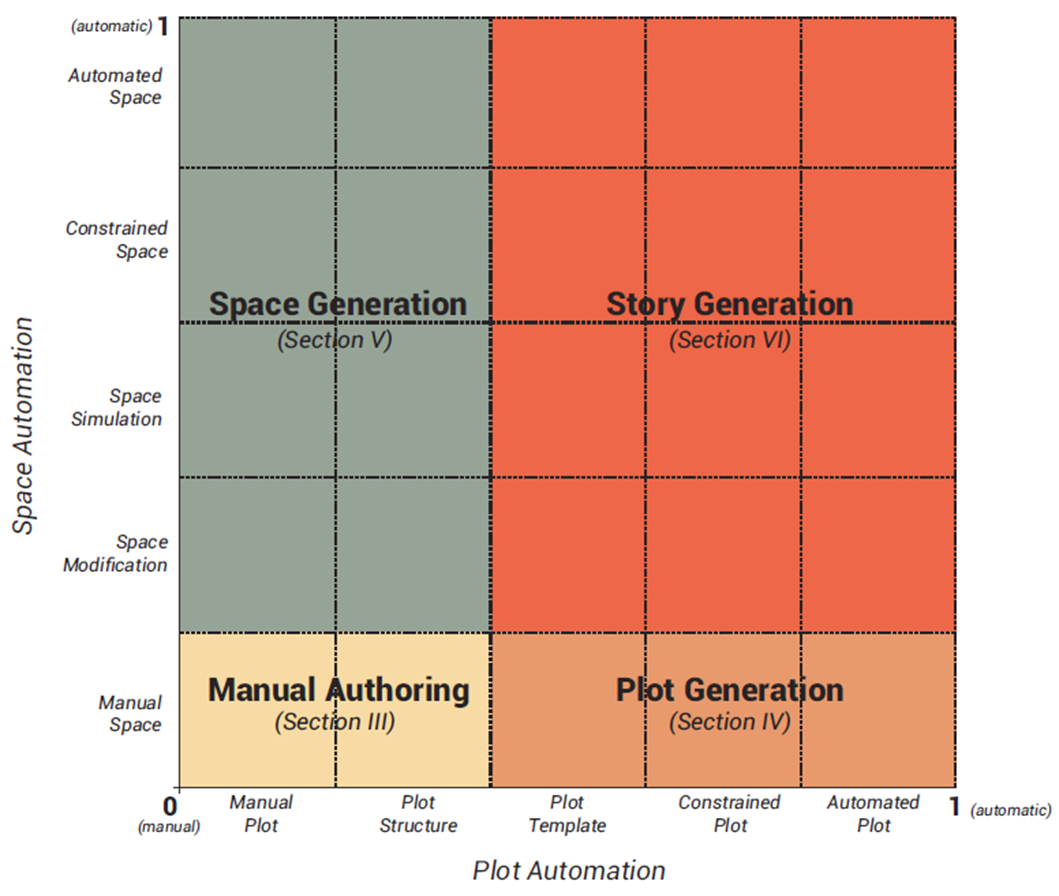
Quest is a task player character(s) can perform to receive reward range from money to the advance of the story. Quest can be categorized into 2 groups, main-quest and side-quest. Main-Quest is quest that revolve around the ‘main’ storyline of the game, while side-quest usually revolve around sub-plot of the story or something completely non-related to the story at all.

**2.2 Procedural Content Generation (PCG)**

Procedural Content Generation (PCG) is a system within a game which creates content for player consumption automatically as the player play the game. The content can be generated by selecting a static pre-determined object from a set, or combining multiple piece of objects together to create a more unique content, and so on. The content can range from new item, new enemy, new non-player character (NPC), new ability to new area. When a player opens a box that is designed to contain random weapon, and gets some weapon that is now a fixed weapon, is an example of PCG.

PCG is implemented into a game for multiple reasons. PCG allow the game to create a ‘unique’ experience for each individual player using the random nature of PCG, thus increase the replayability of the game and value of the game. PCG also reduce the burden of the game developer by lowering the amount of manual labour the developer has to perform to deliver the same amount of content when PCG is not used.

Procedural Quest Generation (PQG) is a subset of PCG and can be generalized into 2 categories, space and plot automation. Space is the game world object and environment, including item, NPC, geometry of the game world, weather, and such. Low space automation may result in only variety in enemy placement or randomness in reward from treasure chest. While high automation may generate a whole city with random NPC for the player to explore and perform task to complete a quest. Plot is the non-tangible part of content which dictate how the player and the tangible part interact with each other. Using a assassination event in a storyline as an example. Low plot automation may randomized only the place the murder happen, medium automation may vary the way the assassination is committed, but the assassination happen nonetheless. However, high automation, and high flexibility in a sense, may even result in an event that the victim actually survive and change the story that happen afterward.



The level of each type of automation, space and plot, can be used to identify the type of quest that could be generated from the PQG system. …………………….[talk about 4 type above picture]

|  |  |
| --- | --- |
| **Space Generation** | **Story Generation** |
| Neo Scavenger  Spelunky | Dwarf Fortress  Rimworld  Pokemon Mystery Dungeon |
| **Manual Authoring** | **Plot Generation** |
| The Witcher Series  Final Fantasy Series  World Of Warcraft | Mount&Blade Series  Sid Meier's Pirates  The Guild Series  The Elder Scroll:Skyrim  Fallout 3, Fallout: New Vegas, Fallout 4 |

[Also put picture of these game up there too.]

From the Figure XXX, Notable examples of PQG are “The Elder Scroll: Skyrim” and “Fallout series”, which use ‘Radiant AI’ from Bethesda Softworks. Both The Elder Scroll and Fallout have a static game world where most object are hand crafted and placed. The Radiant AI system is a PQG system which generate quest for player. The generated quest will have fixed task and narrative, but the object that the player have to interact to complete the quest will be randomly chosen from the object in the game world.………………

In order to create more variety and unique experience for each individual player, procedural quest generation (PQG) is developed and implement into the game. PQG is a subset of procedural content generation (PCG), a system which create content within game automatically and randomly (non-determined). The main purpose of PCG is to present each player with unique content that randomly change every time the game is played.

Still, current commercial PQG (in the game such as Elder Scroll Skyrim, Fallout 4) generated quest by combining multiple pre-scripted part of quest together to create a new quest. This, while guarantee the generated quest to always be functioning and consistent, still limit the flexibility of player action and quest content. Therefore new way to generate quest for computer game are being researched and developed. One such work is the structural analysis by Doran and Parberry [2011]. Doran and Parberry analysis quest from 4 MMORPG games and conclude common structure of quest which can be used in quest generating.

The structural analysis is a quest structure rule which can be used to construct quest.

**2.3 Structural analysis of quest**

Structural Analysis approach in quest generation is a way to construct quest in similar approach to constructing a sentence using ‘common’ grammar. The ‘grammar’ and ‘vocabulary’ rule of structural analysis was created by classification, analysing, and dissecting quests from multiple RPGs game to get a common pattern which all quest shared. In structural analysis approach, quests had been generalized into ‘motivation’, the distinct underlying drive (narrative) that compel the quest. Then within each ‘motivation’, the quest could be categorized into different ‘strategy’, the outline on how the quest (motivation) can be complete (satisfy). And finally, each ‘strategy’ could be linked to specific set of ‘Sequence of Actions’ which describe the general task (action) the player or NPC can to perform to complete the quest. The task (actions) are usually in the <ACTION> form, which can be **breakdowned** into specific ACTION depended on the **Action Rule** table.

**[Figure XAW] show example of <RuleSet> / Action Rule**

Figure XAW show an Action Rule Table from Doran and Parberry [2011],

**2.4 John Grey and Joanna Bryson’s Agent Interaction in Role-Playing-Games \* [CHANGE TITLE TO SOMETHING LIKE… “SIMULATING DYNAMIC NPC INTEREACTION” ]**

……………….

[CHECK PAPER NAME IF FIT BELOW DESCRIBTION]

\*This paper should be about using quest as a way for NPC to interact with each other.

\*EX. A hate B, so A got 'quest to kill B' and perform it.

**……………….**

Quest are not exclusive to player only, NPC can also questing. Usually this kind of NPC-able Quest system are implemented in computer game to increase the level of dynamic environment and interaction of the game world.

Game can implement quest as a way to simulate NPC behaviour to be more realistic and dynamic. A game can create NPC only quest such as ‘hunting bear quest’ for NPC who perform hunter job in the game world. Instead of spawning bear meat directly into that NPC inventory, the NPC has to actually complete the hunting bear quest in order to obtain deer meat.

[ACCTUALLY, HOW IS THIS DIFFERENT FROM NORMAL SCRIPTING?]

[ANSWER: Universal AI that can find way to perform Quest instead of manually scripting every NPC?]

A good example of player sharing quest pool with NPCs can be found in the original build of S.T.A.L.K.E.R, which is now known as S.T.A.L.K.E.R.: Shadow of Chernobyl. In that original build, every NPCs have the same set of available action similar to that of player, along with AI system that allow the NPCs to perform those action accordingly. This system was sound in when look into the game story, the player is one of the many S.T.A.L.K.E.R. in the area, and anyone can be ‘THE ONE’ who solve the mystery that litter around the game world. This system allow the player to encounter different ‘story’ depended on how the player and NPC chose to do what quests in what order. However, the GSC Game World (developer of S.T.A.L.K.E.R.) scrap the system because it is not a ‘fun’ experience for player. Testing players found that they were locked out of content and quests because other NPCs had already finished the quest. And most importantly, NPCs can finish the main story quest before the player and end the game prematurely. This is one of the problem when dynamic Questing NPCs system is used in game which has limited set of quests.

**2.5 A Parametric Analysis and Classification of Quests in MMORPGs**

G

**2.6 แบบจำลอง ความเชื่อ ความต้องการและเจตนา (Belief-Desire-Intention model or BDI)**

**3. Related Work**

Related work in quest generation are consisted of [Role-playing Games], [] , [].

**3.1 Doran and Parberry’s Structure Analysis [2011].**

Structural Analysis approach in quest generation is a way to construct quest in similar approach to constructing a sentence using ‘common’ grammar. The ‘grammar’ and ‘vocabulary’ rule of structural analysis was created by classification, analysing, and dissecting quests from multiple RPGs game to get a common pattern which all quest shared. In structural analysis approach, quests had been generalized into ‘motivation’, the distinct underlying drive (narrative) that compel the quest. Then within each ‘motivation’, the quest could be categorized into different ‘strategy’, the outline on how the quest (motivation) can be complete (satisfy). And finally, each ‘strategy’ could be linked to specific set of ‘Sequence of Actions’ which describe the general task (action) the player or NPC can to perform to complete the quest. The task (actions) are usually in the <ACTION> form, which can be **breakdowned** into specific ACTION depended on the **Action Rule** table.

**[Figure XAW] show example of <RuleSet> / Action Rule**

Figure XAW show an Action Rule Table from Doran and Parberry [2011],

**3.2 Quest Patterns for Story-Based Computer Games**

f

NPC

**3.3 Analysis of ReGEN as a Graph Rewriting System for Quest Generation**

f

ReGEN anaylse

**3.4 Hierarchical Generation of Dynamic and Nondeterministic Quests in Games**

f

**3.5 Natural Language Generation for descriptive texts in interactive games**

f

**3.6 Lee and Cho’s [2012] Dynamic Quest Plot Generation using Petri Net Planning.**

In this work, quest is defined as a sequence of event that happen to form narrative of the quest. the quest generation create quest by chaining multiple ‘events’ together.

**4. ด**

This thesis will use….

In term of programming, the system will be consisted of 2 languages.

1. Prolog. Prolog language is chosen because of its ability to query and backtrack.
2. Java.

----------

The system will be consisted of 3 main components.

1. Prolog query system
2. Eclipse java project which will generated quest structure from ruleset and store all possible path to complete the quest.
3. Ruleset which will dictate on how the quest will be generated and structured.

---------

The quest will be consisted of the following hierarchy

1. QuestFrame
2. Template
3. Component & RuleSet
4. GameState

MAKE SURE TO FIX <rule> AND RuleSet FIX that sometime got used wrongly and cause confusion

MAKE SURE TO FIX <rule> AND RuleSet FIX that sometime got used wrongly and cause confusion

MAKE SURE TO FIX <rule> AND RuleSet FIX that sometime got used wrongly and cause confusion

**[QuestFrame]:**

[QuestFrame] will be the main Framework of the quest. It will contain a List of Template and …..

**Template:**

Template can be described as an ‘arc’ of the quest. Each Template will contain component(s) which represent different states of the quest. A Template can be considered as a ‘mini quest’ if they are by themselves. **Restriction in the size of each Template is implemented to prevented overlong quest. The size can be considered as the number of Components within the Template.**

**<Component>**

<Component> is the smallest part of the pre-determined quest structure before the quest detail is constructed. <Component> can be broken down into either another <Component> or finite Component that can no longer break down further. **For example, <steal> can be broken down into [<goto> stealth take] or [<goto> <kill> take]. Here, ‘stealth’ and ‘take’ is the finite Component that cannot be broken down further.**

**Component:**

In this thesis, Component can be considered as an Doran and Parberry [2011]’sAtom Action. Component represents the changing in game world through the player action that has to be met for the quest to move forward and finally completed. Each component will contain “Start State” and “Goal State”. Start State can be described as the condition that the game world must satisfy at the point the component will be initiated as a part of quest. Goal State is the condition of the game world that must be satisfied for that part of the quest to be considered as complete and player can advance to the next part of the quest.

**Compare to Doran and Parberry [2011], Component can be consider as the [2011] RuleSet. Since the path/sequence of action by player which used to be determined by breaking down RuleSet is REPLACE by the new ‘loop’ system.**

**EACH COMPONENT IN THIS SYSTEM CAN BE CONSIDER AS <Atom Action>**

**INITIAL COMPONENT = <RULE>**

**BREAKDOWN COMPONENT = <ACTION>**

**GameState:**

GameState is the state of the game world. GameState is the collection of multiple ‘Game World Condition’.

GameState used in Component can be categorized into 2 types.

1. Start State: Start State is the requirement condition that the game world must satisfy in order to **start the component**
2. Goal State: Goal State is the game state that when met, that section of the quest is considered completed and the next section can be started, advancing the quest.

GameState used in …………..………………………

………………………………

………………………………

………………………………

**Game World Condition:**

An individual condition or happening in the game world. All Game World Condition in the current game world will be called “GameState”.

---------------------------------------

**NOTE: BELOW RULESET BREAKDOWN IS DIFFERENT FROM THE ORIGINAL SINCE THE new stystem ‘loop’.**

**Original Ruleset:**

Ruleset will be used to determine how <atom> will be generated from <rule>. This system will use the Ruleset from Machado, Santos and Dias’ [2017] work which extended from Doran and Parberry [2011].

**When <rule> is break downed, it should breakdown according to the RuleSet that dictate how <atom action> can be generated from <rule>. For the generated quest to have consistence task and content that fit the narrative of the quest, Doran and Parberry [2011] analysed multiple quests from 4 MMORPGs game to ………………… FIX THIS**

Each <atom action> can be descript as ‘task’ / ‘action’ that the player have to perform to advance the quest. Each <atom action> can be summarized to have ‘input Game State’ and ‘output Game State’. **As shown in Figure AXXA**

**[Figure AXXA]**

When the quest is generated in the previous system, Prolog will be used to breakdown all possible path / sequence of <atom action> from the initial <rule>. Then a path will be chosen and delivered as a quest. This path / sequence can be described as the sequence of <atom action> that could change the starting Game State to desired Game State that satisfy the quest goal. When the player perform the action descript within the <atom action>, the current Game State (input Game State) will change to ‘output Game State’ and so on until the Game State that satisfy the quest ending condition is met.

**NEW RULESET and Usage:**

In this system, the breaking down of <rule> into <atom action> using RuleSet is obsolete by the new system that can generated all possible path / sequence of actions that the player can take to advance from the initial Game State (‘input Game State’ in this case) to the desired Game State (‘output Game State’ in this case).

The new usage of the RuleSet is to prevent the ‘loop’ from achieving certain Game State status before another one that would result in **conflicting** quest, such as reporting the capture of NPC N before NPC N is actually capture.

To achieve this, the new usage of RuleSet will be to generate a sequence of Game State rather than a sequence of action. And instead of generating the sequence from start to finish (Prolog from root to all leafs), the new RuleSet will be used to generated these Game State from finish (goal Game State) to the starting Game State. **For example, <steal> can be broken down into [<goto> stealth take] or [<goto> <kill> take] where the <Rule> can be expanded further.**

**EACH COMPONENT CAN BE CONSIDER AS <Atom Action>**

**INITIAL COMPONENT = <RULE>**

**BREAKDOWN COMPONENT = <ACTION>**

When the <rule> is breakdown in this system from the Template and Component according to RuleSet, the ‘output Game State’ of the last <atom action> will be account for. The system will look at the last <atom action> and determined what’s the ‘output Game State’ of that last <atom action> could be, then determine what the ‘input Game State’ should be so that the last <atom action>’s ‘output game State’ is possible. For example, if the <atom action> is <deliver item A1 to NPC 2B>; then the ‘output Game State’ of this <atom action> is “NPC 2B possess item A1”. Thus, the ‘input Game State’ of thie <atom action> should be “NPC 2B NOT possess item A1”.**As shown in Figure AXXA**

**[Figure AXXA]** (figure of whole ruleset, last one analysis “start” condition)

Then when the next <atom action> is analysed (2nd last <atom action> in this case), the Game State “NPC 2B NOT possess item A1” will be passed down as an additional Game State. **As shown in Figure AXXA**

**[Figure AXXA]** (figure of whole ruleset, 2nd last one analysis “end” condition / AKA: condition required to be met after the <action> is done)

After the analysis , the result would be….. **As shown in Figure AXXA**

**[Figure AXXA]** (figure of whole ruleset, 2nd last one analysis “start” condition / AKA: condition required to be met before the <action> canstart)

When the process exhaust all <atom action> and reach the first ‘input Game State’ of the first <atom action>, the collected Game World Condition (called **Restriction State** from now on) will be used to prevent the system from choosing conflicting action between each Game State. Now when the system start to generate path from the root Game State (start State of the first Component) to the goal state of the first component, the system will check if the Game State Condition generated by these paths conflict with the Restriction State.

For example,

**As shown in Figure AXXA-AXXZ**

**[Figure AXXA]** (figure of [step by step on how the **Restriction State** cancel out path that conflict] 1st step.

**-**

**[Figure AXXZ]** (figure of [step by step] last step.

By doing this, the system can prevent conflict and inconsistence sequence of action. The result path from each Game State to next Game State would retain the consistence of the original RuleSet, while flexible enough to take advantage of the new path generation system.

**NEW RULESET and Usage 2:**

On the other hand, the system can disable RuleSet completely when generating quest path. <RuleSet> that is yet to be broken down into <Atom Action> in the Component can be ignored so that the system would have less restriction in the creating of path.

In this type of configuration, only the ‘Start State’ of the Component will be ……

Similar configuration (disable of using any **<RuelSet> to create Restriction State**) can also be used at Template level……….

**\*\*\*CURRENT CONFIG:**

**Restrucition State will be apply only in component that were created in the same template only.**

**When the RULESET reach the start of the template, and will go into the end of the previous template, all Restrucition State will be reset.**

**THIS SETTING CAN CHANGE TO SEE HOW IT EFFECT THE QUSET GENERATION, BUT FOR DEFAULT SETTING THIS WILL DO.**

***‘Breakdown’ of <Components> into Components***

1. ***According to the <RuleSet> from [2017]***
2. ***According to the***

***‘Breakdown’ of <Components> in the same Template***

***f***

***‘Breakdown’ of <Components> in the different Template***

***f***

**Since there’s a rule governing the connection between Component between these elements already, theese instruction is redundant. All component can be viewed as a long list of Component form single template, no matter how many template they actually generated from.**

This is further restricted in the original RuleSet breakdown by manually selecting <RuleSet> in each “quest section” (Template in this case). The <RuleSet> are put in certain order and type to make sure that no messy or convoluted set of <atom action> is generated.

However, with the new ‘path generation’ of this system, such manual restriction may not serve the system well compare to the previous system. Therefore, it may be more beneficial

**Methodology**

Linking between Template

Each type of [QuestFrame] will have pre-determined set of Template. These are manual

Linking between Component within same Template

1. 1st Component’s Goal State must NOT conflict with 2nd Component’s Start State.
2. 1st Component’s Goal State SHOULD match with 2nd Component’s Start State. But not MANDATORY

Linking between Component across Template

1. 1st Template last Component’s Goal State must NOT conflict with 2nd Template first Component’s Start State.

Quest Generation

[Explain from JAVA]

The program will be developed in Java language using Eclipse along with Prolog plug-in. The Eclipse part of the program will be used for generating framework of quests. Then when the outline of a quest is completed, the detail path and action the player can/have to perform in order to complete the quest will be generated by Prolog. Prolog ability to search for all possible path and back-track are the reasons why Prolog is used to check all of the possible path of the quest, including the feasibility of the quest, if it is completable by the player or not. The user will interact with the program only through the Eclipse.

When the user generates a quest in Eclipse. The user will run QuestGenerator.java and the program will start with initiating [QuestFrame] object which will contain all of the generating quest element. User can determine what kind of quest will be generated by augmenting the QuestFrame setup and quest type.

Then a list of Template object will be initiated within the [QuestFrame]. The Templates within the list will be constructed according to the quest type and settings. The [QuestFrame] can contain multiple Templates within itself and each Templates within the list can be considered as an ‘arc’ within the quest narrative. The number of Template is determined by the configuration of that quest. Short quest may contain only 1 Template, while longer and more complex quest can contain multiple Templates in the Template list.

Then within each Template object, a list of <Component> will be created according to the selected Template setting. Each Template has their own set of <Component> that they always start with. The <Component> will then be broken down into Component until no more <Component> is left. There will be a separated java class that receive list of <Component> and return list of ComponentThe Component will only be represented as string and not actual object. The Components object will be store within each Template that hold the originating <Component> as a separated object class to <Component> class.

**[Figure AQW] this is figure that show the current status of the [QuestFrame] and quest after all above step is done**

**[Figure AQW] this is figure that show the current status of the <Component> is breakdown into Component**

After the list of Components is created, the quest outline can be considered complete. The next step is to fill in detail, narrative, and specific condition of the quest for it to be playable. Due to the changing in the system, RuleSet table no longer represent the direct task that the player has to perform. Therefore if the system is to be able to determined how player can complete the quest using the set of possible action, the Eclipse will query the quest outline to Prolog. Then Prolog will exhaust all possible path of actions player can perform and follow to complete each part of the quest and finally complete the quest.

However, the current quest outline still lack the necessary information required for a thorough query. The current list of Components and their Game State are still lacking in game condition from **Restriction State**. The addition game conditions from Restruction State will **Restriction State** prevent Prolog from creating conflicting sequence of player character action.

In order to obtain **Restriction State,** allcomponents within the quest will have their StartState and GoalState breakdown into a **list** of GameState from all components (AKA NEW <RULE>). Then the system will start to read the list from **back to front, while assembling the necessary GameState that must be met if the GameState is to be achievable from GameState that come before. [เขียนแก้ให้เข้าใจได้ง่ายขึ้น]**

After the **Restriction State** is created, the **[quest generating system]** will be able to determine what condition is prohibited during the ‘path’ generating in order to not create a conflicting and inconsistent action path. This is done by merging the list of **Restriction State** with the Start State and Goal State of Components. The list of **Restriction State** and list of broken down Components should has equal length. After this process is finished, the new list of start and goal Game State would be created. For clear documentation, this list will be called “**Full Conditioning State**”.

After “**Full Conditioning State**” is created. The class QuestGenerator.java will query Prolog and receive all possible paths of action that player can perform to complete the query quest. The following information will be passed to Prolog from Java side of the program.

1. Current Game State: All current game conditions that are store within the system.

2. Quest Outline: The type of quest that is being generated and the list of Template.

3. “**Full Conditioning State”** (first Game State in this case)**:** The list of Start State and Goal State from starting point of the quest to the final point where the quest is completed by the player. Only the next ‘Goal State’ will be sent to Prolog.

4. (If exist) **“Previous Path”**: List of action the player did previously. Will be explained in detail in the following step.

In Prolog, the ”[Precedence?]” will receive all of the mentioned data and try to discover all possible path from the start of the quest to the end of the quest.

The ”[Precedence?]”, called **“QuestPathFinder”**, is the main [Precedence?] that all other loop and [Precedence?] will point to when they had done their query.

**QuestPathFinder**(AllItem,AllCharacter,AllRelation,AllAttribute,character(Name,Level,Status,Attribute,Location),Path,LoopCounter) :-

(

testGameStateKillQuest1(AllItem,AllCharacter,AllRelation,AllAttribute,character(Name,Level,Status,Attribute,Location),Path,LoopCounter) -> true;

testGameStateKillQuest2(AllItem,AllCharacter,AllRelation,AllAttribute,character(Name,Level,Status,Attribute,Location),Path,LoopCounter)

).

**[Figure AQW] Figure of Prolog code……. [REAL VERSION USE REAL PHOTO, also fix all these plachholder naming]**

The first [testGameStateKillQuest1] will point to [Precedence?] that check whether the current game state within the query reach the goal state or not. If it does, the whole query will return true and write all of the Game State and existing Path to text file, which will be read by the Eclipse part of the program for further procedure. Otherwise, the query will be pointed to [testGameStateKillQuest2] for the next action to be added to the path.

When Prolog query into [testGameStateKillQuest2], there are multiple[Precedence?] with the same name as [testGameStateKillQuest2]. Each [testGameStateKillQuest2] will query using different [Precedence?] that represent player action such as “attack(AllItem,AllCharacter,AllRelation,…...,Path,LoopCounter)” and “goto(AllItem,AllCharacter,AllRelation,…...,Path,LoopCounter)”. When these query are finished, it will also add ‘pending resolve’ into the **‘pending list’. ‘pending list’** is used for telling the Prolog system how the game world (Game State) will react (change) to the player action (represent by the recent query). The “Path” atom will also got appended by a string that represent the action taken, such as [attack|Rest\_Of\_Path]. Then it will query further down into the [RESOLVE Precedence] where all of the pending object within **‘pending list’** will be queried and change the current Game Sate that is within the query. In the [ RESOLVE Precedence], the query will keep query back to itself until no **‘pending list’** is left. Finally the query will point back to **QuestPathFinder** for the next loop until the Goal State of this quest is met.

After Prolog has discovered all possible path to the queried Goal State, it will create text file that contain all of the following information in the following format. And also notify Eclipse that the query had been completed.

1st path: ([Game State] Information) , ([Action Path] Information)

2nd path: ([Game State] Information) , ([Action Path] Information)

3rd path: ([Game State] Information) , ([Action Path] Information)

.

….

**[Figure AQW] Figure format and information [MAKE THIS PICTURE]**

After receiving the notification, Eclipse part of the program will read the text file and store all of the information for the next procedure. The next Game State in the “**Full Conditioning State”** will be selected. Then the Game State will be submitted to Prolog for query along with the selected path information. The path will provide ‘current Game State’ information to the query. The “Path” in the query will also start with [Action Path] instead of empty list in order to track the action player has to perform in order to reach the desired Game State.

The Eclipse will query all possible paths from the previous query before moving on to the next Game State in the “**Full Conditioning State”**. These query process will be completed when the last Game State, which represent the quest completion condition, in “**Full Conditioning State”** had been queried. After this point, Prolog will no longer be used within the program.

After all possible paths had been discovered by Prolog and recorded into Eclipse, the program will analyses the quest and evaluate it using similar procedure as of ReGEN’s quest quality measurement. For example, the ‘flexibility’ , or how many approaches/possible paths are there, to finished the quest is measured by the number of possible paths discovered by Prolog and how many common list of actions these path share. After the analysing is completed, the QuestGenerator.java will export the quest into a text file, including the information regarding all possible paths and the analysing of quest quality.

---------

**Expected Outcome**

The system can generate quest according to RuleSets and their conditions.

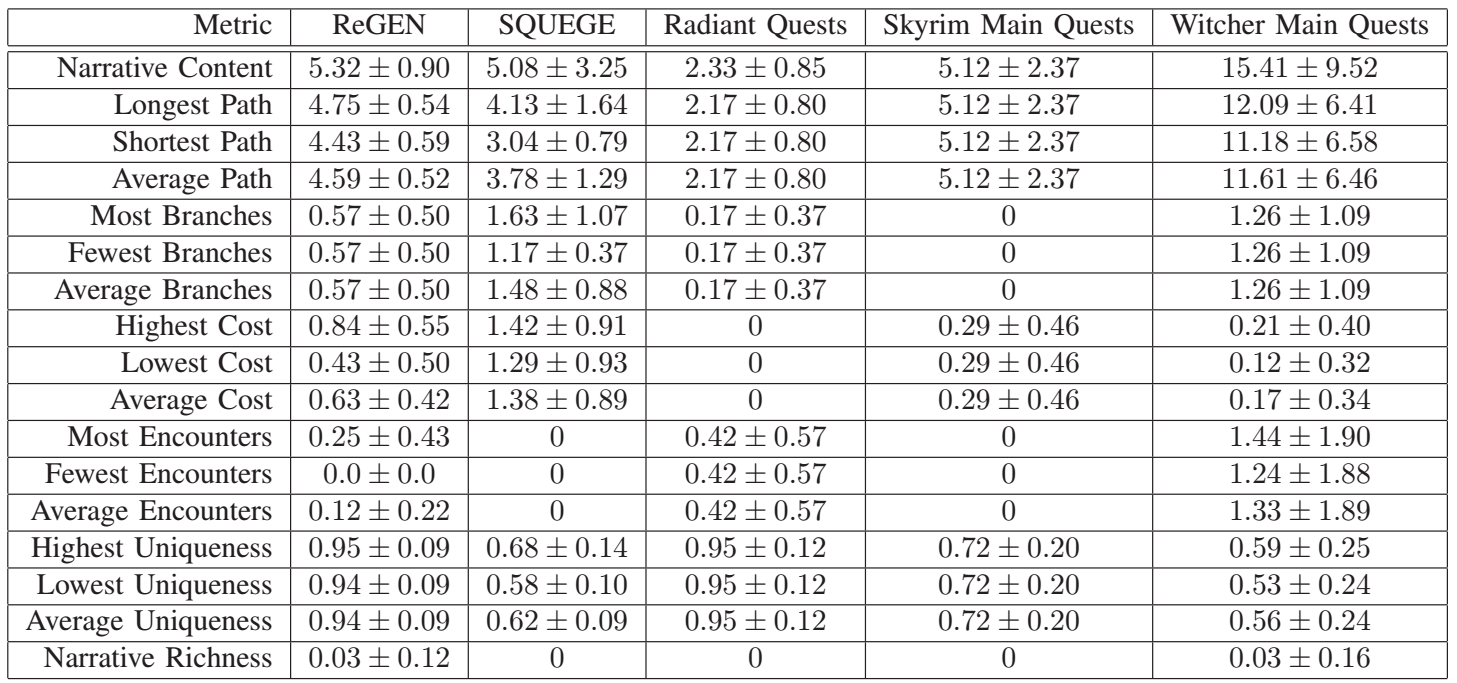
The generated quest can be completed by player character.

The generated quest can be

T

**Measurement**

This thesis will use ReGEN’s metric system. [REFERENCE]

**REGEN**

Below = new measurement / change in how original measure quality.

**Number of Approach**

Branch is originally used for measuring the number of time player is given a choice within the narrative. However, the choice in this place represent player ability to choose the ‘goal state’ condition of a quest rather than player ability to choose how to reach the goal state. Thus it would be more accurate to use a new metric to represent player ‘flexibility’ to reach a goal state.

**Approach will be measured by how many paths of actions player can follow to complete a quest.**

**BELOW = mathematical COUNTING**

**-**

**-Path which share ‘IDENTICAL GAME STATE’ WILL BE COUNTED AS ONLY 1 (IF THE WHOLE PATH SHARE ACTION, even the action is not เรียงเหมือนกัน) ??? [STILL NOT CERTAIN]**

**-**

In addition, **Branch** will not be used in conjunction with **Approach for the whole quest.** The reason is that different ‘branch’ of a quest has different set of ‘goal state’. **Approach** can be used to measure the quest from start until the narrative of the quest reach a ‘choosing’ point. Then each different ‘branches’ must have their own **Approach** metric. If these ‘branches’ shared their **Approach**, a ‘branch’ which has linear/single path of ‘action’ to finish the quest in one ‘goal state’ may has high **Number of Approach** if the other ‘branch’ has multiple paths to finish the quest in different ‘goal state’.

^THIS IS SIMILAR TO COUNTING “MOST / LEAST / AVG ENCOUNTER” [STILL NOT CERTAIN]

**Number of Branches**

We take the number of branches as being the number of times within a narrative that the player may experience an event that leads to two or more possible events. As previously explained, the selection of which event to experience is made by the player, and they may select only one of the possible events. We call this selection process player choice. Note that we are not analyzing the total number of paths, rather, we are trying to analyze the number of times within the narrative that the player will be given a choice, depending on the path they take through the given narrative. Thus we are aiming to determine, the most, fewest and average number of choices a player could be given within the quest. This is one metric which would not normally be a feature of more traditional narrative analysis, but it is specific to an interactive narrative context. We can view the number of branches in a narrative as a measurement of player freedom and/or narrative complexity.

REMOVE METRIC AND REASON.

Encounters: Encounters

[BELOW = COPY FROM ReGEN]

ย่อข้างล่างคล้าวๆพอ

**Narrative Content**

Our first metric relates simply to narrative length, and the total number of story events. While it is impossible to directly relate story length to how good a story is (such as determining if short stories are better than novels or vice versa), there are certainly extremes on both sides that could negatively impact story quality. For example, a side-quest with the following description: Please go outside and collect one flower may be disappointingly short, while an alternative quest stating Please go outside and collect one hundred flowers may feel much too long. The “One Hundred Flowers” quest could also appear lengthy due to the fact that the player is repeating the same actions for the entire quest, which we will discuss below in the “uniqueness” metric. One measure of narrative length is by simply looking at how many narrative events there are overall, regardless of whether the player will be able to experience all events in a single play-through of the quest. We will dub this basic metric Narrative Content as this is not necessarily a measure of narrative length, but rather of how many potential narrative events could occur within the quest.

**Path Length**

We cannot directly measure the length of a narrative when games allow for branching stories as each branch could be any arbitrary length. This means that the best measure for narrative length would involve looking at both the longest, shortest and average path through a narrative, and the amount of narrative seen by the player when taking either of the extreme paths. As with most metrics, it is controversial to state these directly relate to the narrative quality, but we can say that they are one of many metrics which are important to understanding narrative quality. For example, a large difference between the amount of narrative seen when taking the longest path as opposed to the shortest path may show that the player is missing out on much of the potential narrative content if they take the shorter path. Conversely, if both the longest and shortest path are very short compared to the total overall narrative content, we may be in a position where there are many short branching paths, which can be viewed as either a good or bad feature.

**Number of Branches**

We take the number of branches as being the number of times within a narrative that the player may experience an event that leads to two or more possible events. As previously explained, the selection of which event to experience is made by the player, and they may select only one of the possible events. We call this selection process player choice. Note that we are not analyzing the total number of paths, rather, we are trying to analyze the number of times within the narrative that the player will be given a choice, depending on the path they take through the given narrative. Thus we are aiming to determine, the most, fewest and average number of choices a player could be given within the quest. This is one metric which would not normally be a feature of more traditional narrative analysis, but it is specific to an interactive narrative context. We can view the number of branches in a narrative as a measurement of player freedom and/or narrative complexity.

**Cost**

Cost is a metric closely tied with narrative generation systems. In our system, actions such as murdering a person or destroying an item make irreversible changes to the game world. After a point these actions could lead to states where it is no longer possible to generate any narratives since most of the NPCs in the world had been murdered by a player in previous quests. We therefore suggest that certain actions should have a cost associated with them. This lead us to assign any event in the narrative that resulted in an object being effectively removed from the game world as having a cost of one.

In Skyrim’s Radiant Quest system, this potential cost is bypassed by having most of the targets be procedurally generated [2]. For example, in a murder quest, the game will procedurally generate an arbitrary NPC to be the player’s target. Thus, murdering them does not make any changes to the game world. While this alleviates the concerns of cost, this takes away a feeling of importance from the quest. It is interesting to have quests which make definite and consequential changes, even if at a high level these pose a threat to the size or state of the game world. Cost thus represents an important property of narrative quality when narratives have a meaningful interaction with the game world.

**Highest/Lowest/Average Cost**

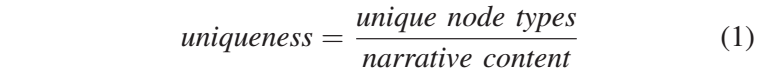
For our analysis of cost, we take a similar approach to the narrative length in that we are interested in the highest, lowest and average possible cost. Again these values let us determine if we are entering any extremes, as well as exploring the variety of possible outcomes based off of player choice. We do not see any purpose in including a metric of Cost Content, similar to Narrative Content as we are not interested in the overall presence of cost events, we are simply interested in how much cost is guaranteed to occur (lowest cost) and how much cost is possible to occur (highest cost).

**Most/Fewest/Average Encounters**

Encounters is a metric which can be considered an alternative to the cost metric. An encounter relates to an exciting event such as a fight with a monster that does not make any irreversible changes to the game world. Essentially, a fight encounter may be the same as a cost action, but whether or not it is an encounter or cost action depends on the target of the event. If the target is renewable, meaning the game may generate an infinite number of said objects, then the action is an encounter action. If the game cannot generate more of a given object, then the target is non-renewable and the corresponding action is a cost action. This metric has the disadvantage that in many of the examined narratives, the game world contained many random monsters placed in the world environment, but fighting them was incidental and not directly represented as part of the quest structure. We retained this metric since there were some quests where certain types of encounters were deemed important to quest progression, such as quests wherein the player must kill X number of creatures to proceed. Once again, we analyze both the most and fewest possible encounters, as well as the average number of encounters.

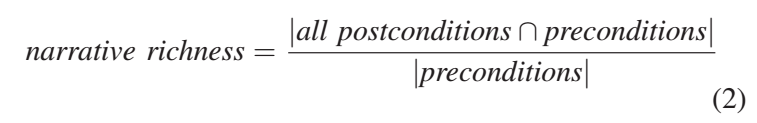
**Highest/Lowest/Average Uniqueness**

As mentioned in the preceding section, for metric analysis we assigned a key action to each event in our story. This we use to help define our uniqueness metric. For example if a node involves murdering someone, then the node is a Murder node. If the node involves travelling to a location, then this is a Go To node, and so on. Using this we can determine how many unique node types there are in a story. A story with two murder nodes, for example, would qualify as having only one unique node type. If we divide the number of unique node types by the narrative content, we get a measure of what percent of our total story nodes are unique, which we choose to call the story’s uniqueness. This is shown in Equation 1



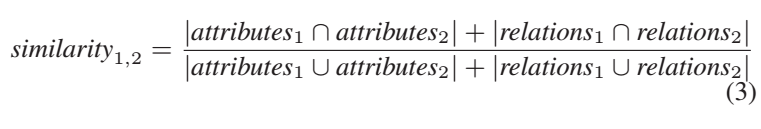
This means that our measure of uniqueness can be considered as an intra-narrative measurement of uniqueness. Referring back to the One Hundred Flowers quest, we can break the quest down into one hundred events, where each event is a COLLECT event. This would yield a narrative content of 100, but a uniqueness of only one percent. Conversely, the One Flower quest would have narrative content of one, but a uniqueness of 100 percent. An example of this in the commercial game Skyrim would be a number of quests within the Civil War chain of side-quests. These quests, usually prefixed with The Battle of followed by the location name, consist entirely of killing a certain number of enemies at different locations. This results in a repetitive experience for the user which regardless of initial interest inevitably becomes dull. We do believe that uniqueness has a direct effect on narrative quality, as it helps to determine whether a narrative faces issues of repetitiveness. Again, while there have been many examples of games that use repetitive actions, these games are often not story based. Future work of this research is to create the inter-narrative equivalent of this metric, allowing us to compare narratives to see how similar one generated narrative is to another. This prevents a case where a narrative might in itself be unique, but many of the same narratives may be occurring in a row. Returning to the Skyrim example, the Battle of side-quests are additionally repetitive in that there are several of them and in no case is there any change in the actions which need to be taken by the player. Once again, since the uniqueness of a quest may change depending on the path the player takes, we look at the highest, lowest and average uniqueness score for each narrative.

**Narrative** Richness  
In an intuitive sense, narrative “richness” or “depth” closely relates to the how surprising or interesting narrative events appear to the player. Plot twists and realization of sub-plots add to perceived complexity and interest, but require the player to experience and even influence narrative events in a way that does not directly relate to the current goal. We interpret this as a metric in terms of the unintentional consequences of a narrative, since it attempts to measure how much of a given narrative has been influenced from past narratives, without being a direct goal of adjoining steps. The richness of a narrative may be influenced by multiple features, but we are currently only evaluating a narrative for richness in terms of these consequences. Future work in this area would aim to formalize more features of narrative richness, and validate it against the opinion of human players, in order to further this particular metric. In our system, we keep a store of all the changes made to the game world by each narrative created. We call these the postconditions of our narrative. We also view the game world conditions for the IRRs and SRRs as potential preconditions. Preconditions can either be satisfied by conditions in the game world which were not the result of previous narratives, and conditions which were. For example, if in a previous narrative the player made two characters hate each other, then if a narrative is generated with the precondition that those two characters hated each other, the hate precondition was only satisfied because of the actions unknowingly taken by the player in the previous narrative. This then allows us to define narrative richness in terms of the percentage of the preconditions for our narrative which were satisfied by the postconditions of any previous narratives, as shown in Equation 2. This metric could be viewed as an inter-narrative metric. CD Projekt RED’s The Witcher is a game known for using this feature, where actions taken by the user in previous quests result in them experiencing different events in later quests. An example of this occurs in an early quest entitled Of Monsters and Men, where the player is given the choice to defend a character accused of witchcraft, or leave her to be killed. In a much later quest, Frozen Reflections, the player encounters the witch. If the player saved the witch, then she is alive and provides the player with potions. If the player left the witch to die, then she is instead a vengeful spirit who attacks the player. Our narrative richness metric aims to capture the concept that the narrative experienced by the player has changed due to seemingly arbitrary choices made by the player in an earlier quest. Note that this metric cannot be evaluated based on a user’s path, as with the above metrics, since richness depends on the impact of a choice on all possible futures—we need to look at how the whole quest itself is the result of previous actions, and not just for an individual user path.



**Weight of Choices**

Our final metric is to examine the effect that choices have on the number of final possible game worlds. As mentioned in the previous section, whenever our stories have a branching event we split our simulation into two parts to represent the two new possible game worlds. We then continue generating narratives from these two new game worlds. After some predetermined number of iterations, we then compare each final game world to each other final game world. This involves comparing each object’s attributes and relations by dividing the number of attributes and relations that are the same between both worlds by the total number of attributes and relations in each game world. This gives the similarity between each game world, as shown in Equation 3. We believe this metric is important since it highlights the importance of the choices made by the player in the game in a quantifiable way.



**IGNORE BELOW**

Most/Fewest/Average Branches: The number of branches show that on average, one in every two of our generated stories will contain at least one branching path. Following our criteria, this falls well below the results of both the SQUEGE narratives and The Witcher narratives, where The Witcher has on average one branching path per narrative and the SQUEGE narratives have closer to two branching paths. The Skyrim main quests are always linear and therefore do not have any branches, whereas the Skyrim Radiant Quests very infrequently contain branching paths. Since our criteria state that a large number of branches indicate better narratives, then the SQUEGE and Witcher narratives are superior in this area to our generated metrics. Since we have made the assumption that The Witcher quests are representations of good narrative structure, we should therefore add more rewrite rules which create branching paths in the narrative in order to compete with its results. This is again one of the strong benefits of defining narrative metrics and comparing our generated narratives to others, since it gives insight into how we can restructure our narrative generation tool in such a way that it produces measurably better narratives.

Highest/Lowest/Average Cost: Comparing costs, we see that our stories contain, on average, at least one potentially irreversible action per narrative. The results for SQUEGE are even higher, with at least one cost unit per narrative regardless of path chosen. The Radiant Quests always have no cost because all destroyed objects are procedurally generated for each quest. While this is an interesting means of having narrative generation with no potential cost to game environment, this takes away a sense of purpose from the side quest, since as a result the quest makes no noticeable difference in the game world. In spite of this, our costs are still high compared to the Skyrim and Witcher main quests, which have costs between 0.1 and 0.3. Following the criteria, aiming to reduce the number of rules which include an irreversible action would improve not necessarily narrative quality, but rather the lifespan of the quest generation process itself.

Most/Fewest/Average Encounters: In examining encounters, we reiterate that in most RPGs, encounters are implied but not explicitly stated in quest structure. For example, in Skyrim, a player encounters many monsters travelling through the game world, but these are random encounters and not stated in the quest description. SQUEGE, makes no such explicit definitions of encounter either. However, encounters are explicitly defined in our system, The Witcher main quests and the Radiant Quests. We define having more encounters as being a positive, and in this instance our system generates encounter events much less frequently than either the Radiant Quests or The Witcher. Again, this is a feature which can be tweaked with the creation of and/or modification to, the sets of rewrite rules in our system.

Highest/Lowest/Average Uniqueness: One of the more interesting metrics analyzed is that of uniqueness. As mentioned before, this metric is not perfect since we take only the primary action of each narrative event to be a description of that event. This ignores, for example, the implied random encounters in the Skyrim world, and does not account for player preference with regards to narrative content. We still feel however, that provides strong insight into a narrative’s uniqueness as it is the primary event that is usually of most interest and importance to the user. Given that we regard repetitive events as detrimental to narrative quality, this is one metric of which a higher value directly implies an increase in narrative quality

Narrative Richness: We then examined both our narratives and the main quest narrative of The Witcher in terms of our narrative richness metric. The narrative richness metric could really only be evaluated for these two sets of quests since it is only in these two systems that actions taken by the player can have unforeseen consequences in later narratives. Although the Skyrim quests have some indirect effects on the Skyrim world, these changes do not effect the main quest, whereas in The Witcher indirect impact is an important part of player (and narrative) choice [28]. In this case ReGEN results are shown to be approximately the same as the results for The Witcher. What is interesting is that for both systems, the narrative richness is very small. While we have fallen within the metrics given by the “good” narrative, it would be interesting to further explore ways of making generated narratives more dependant on the results of previously generated narratives.

**5. Objective**

This thesis

**5. วัตถุประสงค์ของการวิจัย**

งานวิจัยนี้มีวัตถุประสงค์เพื่อพัฒนาระบบการดำเนินเนื้อเรื่องของเกมประเภทสวมบทบาทที่สามารถปรับเปลี่ยนเนื้อเรื่องตามบุคลิกลักษณะการเล่นของผู้เล่นเพื่อให้เนื้อเรื่องมีความเหมาะสมต่อผู้เล่นและสามารถทำให้ผู้เล่นพึงพอใจได้

**5. Scope of Work**

This thesis