การทำงานของคลาส และ แพ็กเกจ

โปรแกรมทำการแบ่งโมดูลออกเป็น 2 แพ็กเกจได้แก่ parsers และ entities โดยที่ parsers มีหน้าที่แจงคอมไพล์ และดำเนินการ construction plan ของผู้เล่น ส่วน entities จะเก็บข้อมูลต่าง ๆ ภายในตัวเกมสำหรับการนำไปคำนวณใน parsers นอกจากนี้โปรแกรมจะมีคลาสแยกอื่น ๆ อีก สำหรับการควบคุมการทำงานโดยรวมของเกม

Package: “parsers”

Interfaces:

`Node` – ระบุการทำงานของโหนดใน *Abstract Syntax Tree*

`Statement`: `Node` – ระบุการทำงานของโหนดดำเนินการ (Executables)

`Expression: Node` – ระบุการทำงานของโหนดคำนวณค่า (Evaluators)

Classes:

`StatementParser` – แจง (parse) construction plan

`Tokenizer` – แยก construction plan ออกเป็น tokens

`ConstructionPlan: Statement` – รองรับงานทำงานทั้งหมดใน construction plan

`ActionCmd`: `Statement` – รองรับการทำงานคำสั่ง `done` กับ `relocate`

`MoveCmd: Statement` – รองรับการทำงานคำสั่ง `move`

`RegionCmd: Statement` – รองรับการทำงานคำสั่ง `invest` กับ `collect`

`AttackCmd: Statement` – รองรับการทำงานคำสั่ง `shoot`

`AssignStatement: Statement` – รองรับการกำหนดตัวแปร

`BlockStatement: Statement` – เก็บหลาย `Statements` เป็น `Statement` เดียว

`IfStatement: Statement` – ดำเนินการทำงาน *conditional statements*

`WhileStatement: Statement` – ดำเนินการทำงาน *while–loop*

`IntLit: Expression` – เก็บเลขจำนวนเต็ม

`BinaryArithExpr: Expression` – คลาสดำเนินการคำนวณ

`Identifier: Expression` – คืนค่าตัวแปร และ Special variables

`OpponentExpr: Expression` – คำนวณค่าจากคำสั่ง `opponent`

`NearbyExpr: Expression` – คำนวณค่าจากคำสั่ง `nearby`

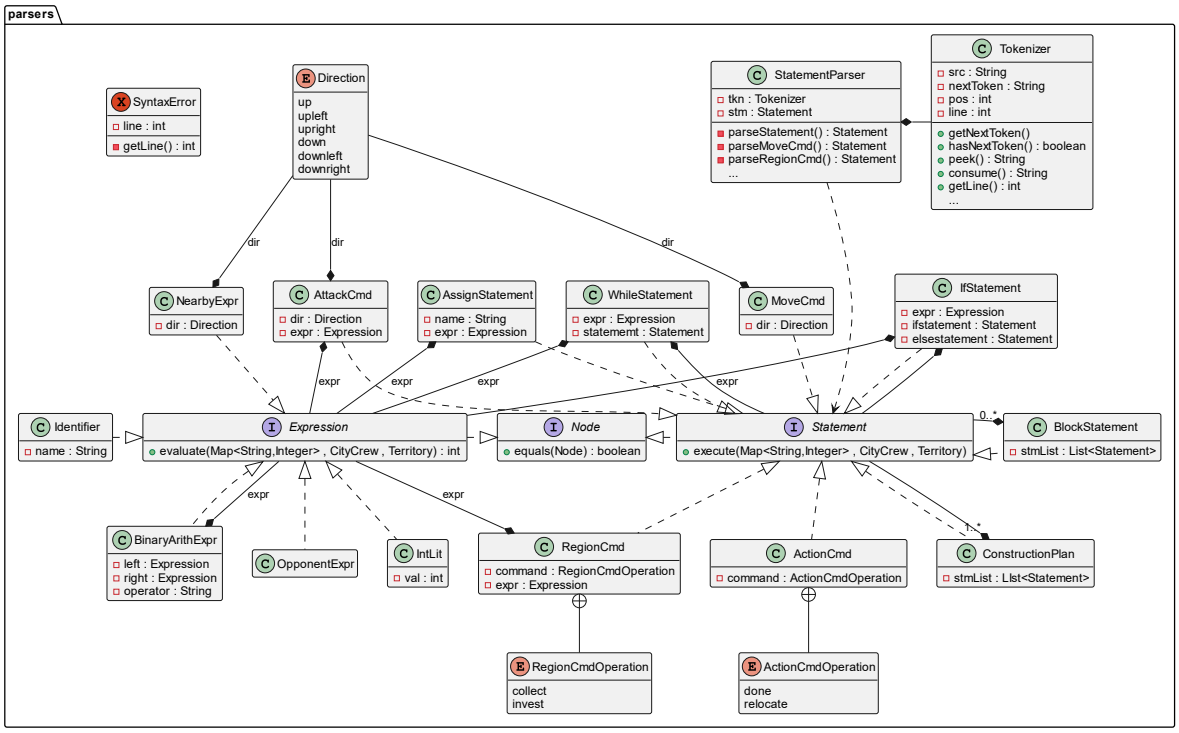
Exception Classes:

`SyntaxError: Exception` – รองรับกรณีที่ construction plan ไม่สามารถ compile ผ่าน หรือ execute แล้วเกิดข้อผิดพลาด

Enumerator:

`Direction` – รองรับข้อมูลประเภททิศทาง

คลาสในแพ็กเกจ **parsers** โดยส่วนมากจะมีหน้าที่เป็นโหนดภายใน *Abstract Syntax Tree* โดยแบ่งแยกคำสั่งต่าง ๆ ลงคลาสโหนดตามพารามิเตอร์ (หรือ *signature*) ที่แต่ละคำสั่งรับ (เช่น **move** ที่รับ **Direction** จะถูกแยกออกจาก **invest** กับ **collect** ที่รับ **Expression**) ซึ่งทำให้นอกสามารถสามารถทำให้ทำงานเร็วขึ้นแล้ว (จากที่ไม่จำเป็นต้องเช็คประเภทของคำสั่งบ่อย) โครงสร้างนี้จะสามารถทำให้แก้ไขโค้ดได้สะดวกขึ้นด้วย ในแต่ละ **Statements** กับ **Expressions** จะสามารถชี้ไปยังโหนดลูกเพื่อรอการทำงานแบบ *Tree* เนื่องจากคลาสจำพวกโหนดจะสามารถส่งผลลัพธ์ได้ออกเป็นสองประเภท ได้แก่ **void** (สำหรับ **Statement**) และ **Long** (สำหรับ **Expression**) จึงแบ่งอินเตอร์เฟสออกเป็น 2 อินเตอร์เฟสเพื่อป้องกันความสับสนขณะพัฒนาตัวโปรแกรม โดยที่คลาสในแพ็กเกจ **parsers** จะสามารถดำเนินการและคำนวณค่าผ่าน พารามิเตอร์ 3 ตัว ได้แก่ **bindings** – แมปชี้ค่าจากชื่อตัวแปรไปยังค่าของตัวแปร, **CityCrew** – ผู้ประกาศใช้คำสั่ง, และ **Territory** – สถานะอื่น ๆ ของตัวเกม โดยที่สองพารามิเตอร์หลังจะเก็บข้อมูลไว้ในคลาสซึ่งอยู่ในแพ็กเกจ entities



Class Diagram of parsers package

ในบางคลาสมีการเก็บ **Statement**, **Expression**, หรือ **Direction** ด้วย ซึ่งจะต้องไม่ใช่ **null** ก่อนที่จะส่งผ่าน constructor ได้ นอกจากนี้ คลาสที่เก็บหลาย ๆ **Statement** เช่น **ConstructionPlan** หรือ **BlockStatement** จะใช้ **Collection (interface)** ในการเก็บเพื่อให้สะดวกสำหรับการวนซ้ำเป็นลำดับชัดเจน

Package: “entities”

Interfaces:

Coordinated – ระบุการทำงานของคลาสที่มีตำแหน่งภายใน territory

Classes:

`Position: Coordinated` – เก็บค่าตำแหน่งใน territory เพื่อที่จะให้โค้ดสามารถส่งทั้งตำแหน่ง `row` กับ `col` พร้อมกันเป็นพารามิเตอร์เดียว

`CityCrew: Coordinated` – เก็บข้อมูลต่าง ๆ ของหนึ่งผู้เล่น

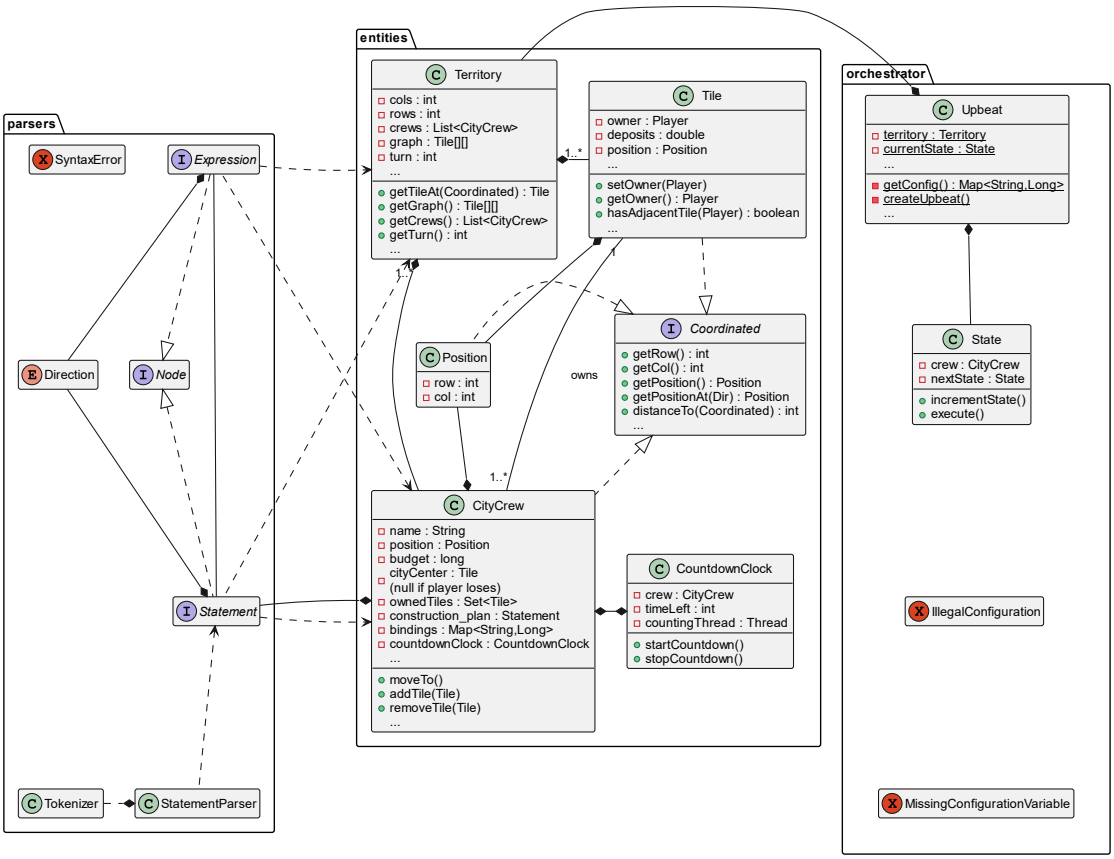
`Tile: Coordinated` – เก็บข้อมูลต่าง ๆ ของหนึ่ง *hexagonal region*

`CountdownClock:` ควบคุมการจับเวลาผู้เล่น

`Territory` – เก็บอ็อบเจกต์ ผู้เล่น, *hexagonal region*, และข้อมูลอื่น ๆ

คลาสในแพ็กเกจ `entities` มีหน้าที่ในการเก็บข้อมูลต่าง ๆ ภายในตัวเกมอย่างเช่น สถานะผู้เล่น สถานะ *region* เป็นต้น เพื่อให้ `Statements` กับ `Expressions` ในแพ็กเกจ `parsers` สามารถคำนวณ ดำเนินการ และเปลี่ยนแปลงข้อมูลในแพ็กเกจนี้ได้ แพ็กเกจ `entities` ใช้อินเตอร์เฟส `Coordinated` อยู่เพียงอินเตอร์เฟสเดียว ซึ่งอินเตอร์เฟสนี้จะมีหน้าที่รองรับเมท็อดคำนวณเกี่ยวข้องกับตำแหน่ง เพื่อให้ง่ายกับการคำนวณผ่านคำสั่งบางคำสั่ง อย่างเช่น `opponent move` หรือ `relocate` เป็นต้น

`CityCrew` จะเก็บอ็อบเจกต์ `Tile` ที่มีอยู่ไว้ใน `Set` และ `Tile` ก็จะเก็บอ็อบเจกต์ `CityCrew` ที่เป็นเจ้าของด้วยเช่นกัน (โดยที่อ็อบเจกต์ `CityCrew` มีค่าเป็น `null` ก็ต่อเมื่อ `Tile` นี้ไม่มีเจ้าของ) ทั้งนี้เพื่อให้การคำนวณใน `parser` สามารถเรียกค่าต่าง ๆ ได้เร็วขึ้น แต่ก็จะส่งผลเสียที่จะต้องเปลี่ยนแปลงหลายตัวแปรหลายจุดเมื่อต้องการเปลี่ยนแปลงความเป็นเจ้าของ นอกจากนี้ `Territory` จะเก็บอ็อบเจกต์ผู้เล่นทุก ๆ คนไว้ในรูปของ `List` เพื่อที่จะได้ระบุลำดับตาของผู้เล่น และ `remove` ผู้เล่นที่แพ้แล้วออกไปได้ง่ายกว่า `Arrays` ส่วน `Tiles` จะเก็บไว้ใน *2-Dimensional Array* จากที่ตัวกลุ่มตัวแปรนี้จะไม่มีการเปลี่ยนแปลงใด ๆ เลย



Simplified Class Diagram of the entire program

Fields ต่าง ๆ ของคลาสใน `entities` ที่เป็นตัวเลขจะไม่เป็นจำนวนลบเสมอ ส่วนอ็อบเจกต์ส่วนมาก เช่น `graph` `crews` `position` ฯลฯ จะต้องไม่เป็น `null`

Package: “orchestrator”

Classes:

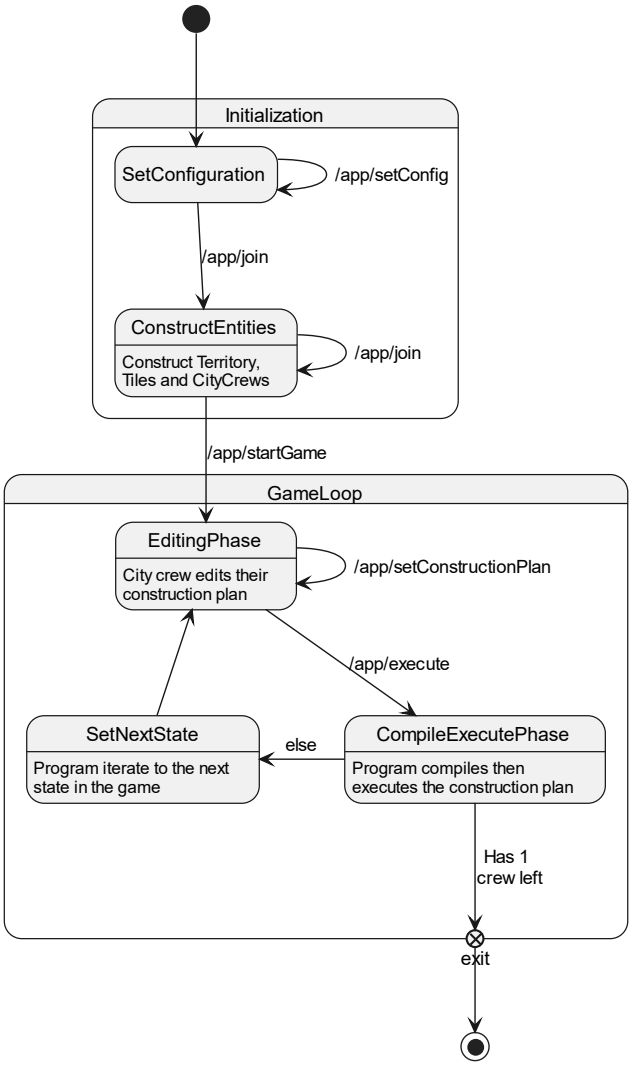
`Upbeat` – ควบคุมการทำงานเป็นขั้นตอน โดยที่ขั้นตอนการทำงานจะระบุไว้เป็น *state machine* ในหน้าถัดไป

`State` – ควบคุมการทำงานต่าง ๆ ภายในตาของผู้เล่นหนึ่งคน

Exception Classes:

`MissingConfigurationVariable: Exception` – รองรับกรณีที่มีข้อมูลจำเพาะ (parameter specification) ขาดหายไปภายใน *Configuration File*

`IllegalConfiguration: Exception` – รองรับกรณีที่มีข้อมูลจำเพาะที่ไม่ถูกต้อง เช่น มีการเก็บค่าจำนวนเต็มลบ หรือ อ่านออกมาแล้วเป็น `null`



State Machine ของโปรแกรม

แผนการทดสอบที่ 1

Testing

– จะทำการ test โดยใช้หลักการของ *black–box testing* โดยจะสนใจแค่ input ที่ใส่เข้าไปในระบบ แล้วทำการเช็คว่า output ที่ออกมานั้นตรงตาม expected result หรือไม่ โดยจะใช้วิธี *equivalence partitioning*

Equivalence partitioning

– จะทำการทดสอบว่า input ที่รับมานั้นอยู่ในขอบเขตหรือไม่ ตัวอย่าง สมมติว่า เรากำหนดให้ค่า `max_deposit` ของแต่ละ region นั้นสามารถเก็บเงินได้สูงสุด คือ 100,000 โดยเราจะแบ่งการรับ input เป็นดังนี้

Equivalence partitioning

Invalid input	valid input	Invalid input
< 0	0–100,000	> 100,000

หากในกรณีของ valid input รับตัวเลขตั้งแต่ 1–100,000 เข้ามาแล้วโปรแกรมจะต้องทำงานต่อได้

ในกรณีของ invalid input หากรับตัวเลขที่น้อยกว่า 0 หรือ มากกว่า 100,000 เข้ามาโปรแกรมก็จะไม่สามารถทำงานต่อได้

โดยระหว่างที่โปรแกรมทำงานมักเกิด exception ต่าง ๆ ขึ้นมากมายไม่ว่าจะเป็น `ArithmeticException` ในกรณีที่หารด้วย 0 , `SyntaxError` ในกรณีที่ `parseExpression` นั้นไม่ถูกต้อง หรือ `NullPointerException` ในกรณีของ `Tile` ซึ่งเก็บข้อมูลต่าง ๆ ในหนึ่ง hexagonal region นั้น หรือหากผู้เล่นคนที่ครอบครองได้ตายลงไป ข้อมูลของผู้เล่นคนนั้นก็จะเป็น `null` ในช่องนั้น เปรียบเสมือนได้ว่า ไม่มีใครครอบครองช่องนั้น โดย exception ที่เกิดนั้นเราก็จะทำการจัดการให้ได้มากที่สุดเท่าที่เป็นไปได้

แผนการทำงานที่ 1

การทำงานจะแบ่งออกเป็น 3 ช่วงเวลาดังนี้

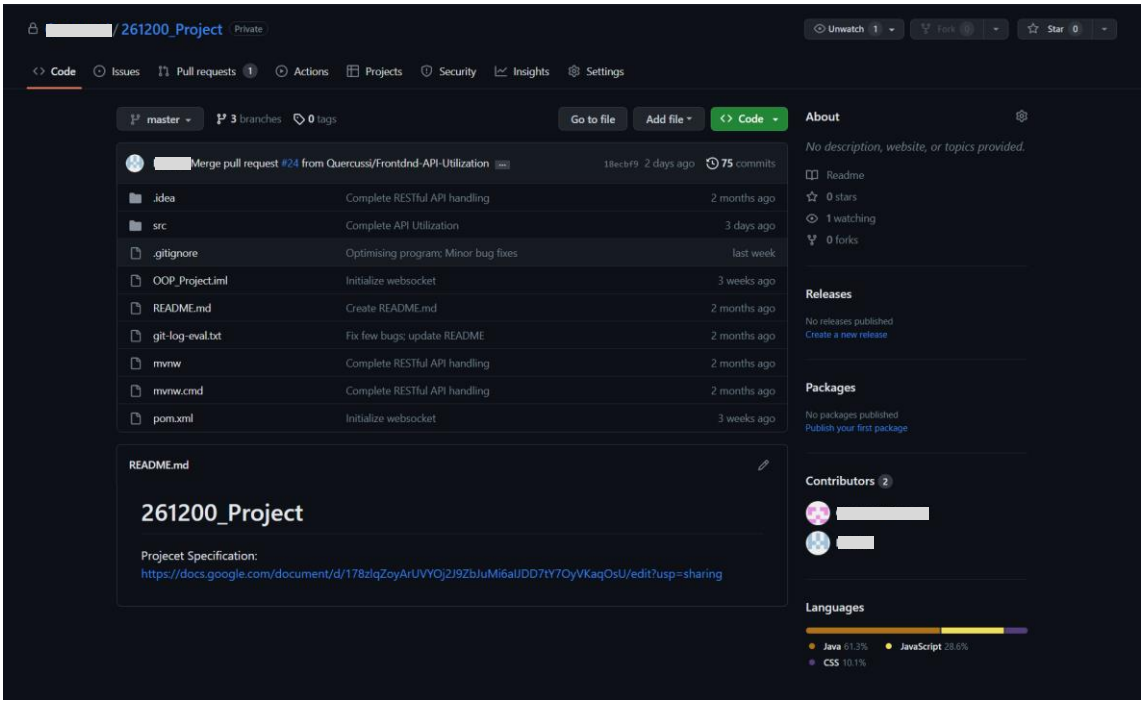
ช่วงการทำงานที่	*รายละเอียดการทำงาน*	*วันเดดไลน์*
1	ทำโค้ดสำหรับการแจง *construction plan* ให้เสร็จ	9/1/2023
2	ทำโค้ดสำหรับแพ็กเกจ `entities` การคำนวณใน `parsers` และ คลาสภายนอกให้เสร็จ	16/1/2023
3	พัฒนาโค้ดให้ดีขึ้น และทำให้ตัวโปรแกรมสามารถเลียนแบบตัวเกมสำเร็จได้	23/1/2023

โดยมีการแบ่งงานดังนี้

640610621	*640610639*
– `Package Parsers` – `Statement: IfStatement WhileStatement BlockStatement` – `Expression:` อื่น ๆ เช่น `IntLit BinaryArithExpr` เป็นต้น – `Tokenizer` กับ `StatementParser` (แจงตามคลาสที่ได้รับมอบหมาย)	– `Statement` คำสั่ง เช่น `RelocateCmd ActionCmd MoveCmd` เป็นต้น – `Expression: Identifier OpponentCmd NearbyCmd` – `Tokenizer` กับ `StatementParser` (แจงตามคลาสที่ได้รับมอบหมาย)
– `Package Entities` – ทำคลาส `CityCrew Tile` และ `Territory`	– *Methods* ของ `Coordinated` (ทำให้เป็น `default`) – ทำคลาส `CityCrew` กับ `Countdown`
– Miscellaneous – ทำ Mock Game	– ทำ Mock Game – ทำคลาส Upbeat

โดยขณะที่พัฒนาโค้ด ทุกคนจะช่วยกันทำ Tester เท่าที่จำเป็น

จากที่โครงสร้างโปรแกรมมีความซับซ้อนและความควบแน่นสูง การพัฒนาอาจจะเป็นไปได้ยากบ้าง แล้วยังจำเป็นต้องแบ่งความพึ่งพาของคลาส *(Key Dependency)* ออกมาเป็นหลายส่วนด้วย พวกเราจึงทำงานด้วยการร่างแปลงของคลาสที่จำเป็นให้ได้ก่อน (โดยเฉพาะส่วนของ *methods* ที่เป็น *public*) โดยยังไม่ระบุขั้นตอนการทำงานให้แน่ชัดคล้ายคลึงกับการประกาศ *prototype*, ตรวจเช็คการสื่อสารระหว่างคลาส, แล้วค่อยเริ่มต้นพัฒนา ทั่งนี้เพื่อให้สมาชิกสามารถแยกย้ายทำงานด้วยตนเองได้



ภาพ Git Repository ของทีม

ปัญหาที่พบและสิ่งที่ได้เรียนรู้จากการพัฒนาที่ 1

สิ่งที่ได้เรียนรู้จากการพัฒนา *parser* กับ *evaluator*

- การทดสอบส่วนของ evaluator ให้ครบถ้วนสมบูรณ์จะเป็นไปได้ยาก แต่กลับพบว่าการพัฒนาและทดสอบส่วนของ parser จะเป็นไปได้ง่ายกว่ามาก จากที่การทำงานของ parser จะไม่ต้องคำนึงถึงพารามิเตอร์มากเท่า evaluator เพียงดูที่ *instance* ต่าง ๆ ของแต่ละโหนดว่าตรงกับโค้ดที่ใส่ลงไปหรือไม่ ก็จะสามารถค้นหาบัคได้แล้ว ต่างจาก evaluator ที่เมื่อเกิด *logical bug* ที จะต้องไล่ดูการเปลี่ยนแปลง instance ต่าง ๆ จากการทำงานของทุก ๆ `Statements` กับ `Expressions` ภายใน *Syntax Tree* แล้วต้องค้นหาว่าบรรทัดใดของโค้ด construction plan และประเภทของคำสั่งที่ส่งผลให้เกิดบัค ก่อนที่จะเริ่มแก้โค้ดได้

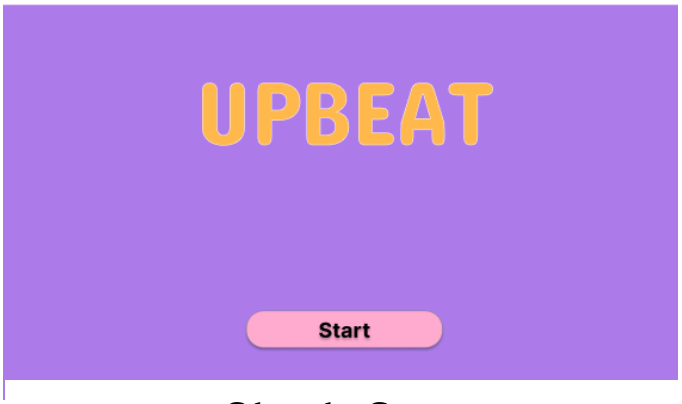
ปัญหาที่พบเจอ

- โปรแกรมสามารถนำไปทดสอบได้ยาก จากการที่มีหลายคำสั่งและกรณีต่าง ๆ มาก เมื่อเกิดบัคทีก็จะต้องใช้เวลานานในการดีบัค นอกจากนี้ตัวเกมก็ยังไม่มี *visualization* ที่ให้เห็นผลการทำงานได้ชัดเจนอีกด้วย ส่งผลให้การพัฒนาเป็นไปได้ล่าช้า และสามารถเกิดช่องโหว่ภายในตัวโปรแกรมโดยไม่รู้ตัวได้
- ขาดการสื่อสารภายในทีมที่มีประสิทธิภาพ เช่น ระบุการทำงานและการสื่อสารระหว่างแต่ละคลาสได้ไม่ละเอียดมากพอ หรือไม่มีการสื่อสารกับทีมเมื่อเกิดความไม่มั่นใจของการทำงาน เป็นต้น ซึ่งทำให้เกิดการตีงานกลับไปแก้บ่อยครั้ง ซึ่งทำให้ช่วงการทำงานจริงล่าช้ามากกว่าแผนการทำงาน (พบช้าสุดอยู่ที่ 5 วัน โดยเกิดในช่วงของการออกแบบ `Package Entities` แต่กลับพบว่าส่วนของการออกแบบ `Package Parsers` จะเป็นไปได้ด้วยดี)
- ช่วงเวลาในการทำงานมีไม่พอ ส่งผลให้ผลงานออกมาแบบไม่สมบูรณ์

ส่วนของโปรเจคที่ยังไม่เสร็จสมบูรณ์

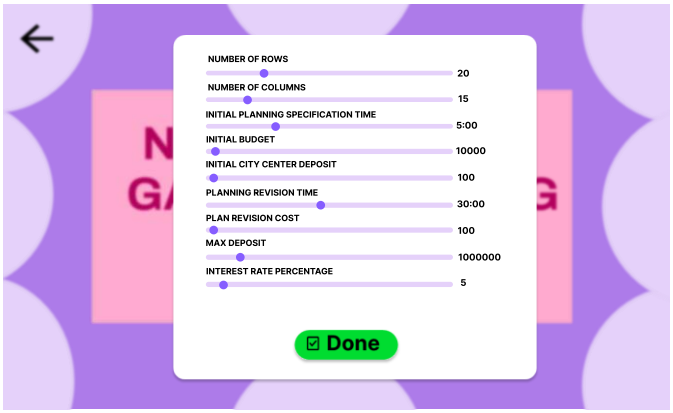
- ไม่ได้สร้างคลาส `Countdown` ที่จับเวลาผู้เล่น เนื่องจากยังไม่มีตัวเกมสมบูรณ์ และไม่มีช่องทางในการเขียนโค้ดสำหรับผู้เล่น

UI Design

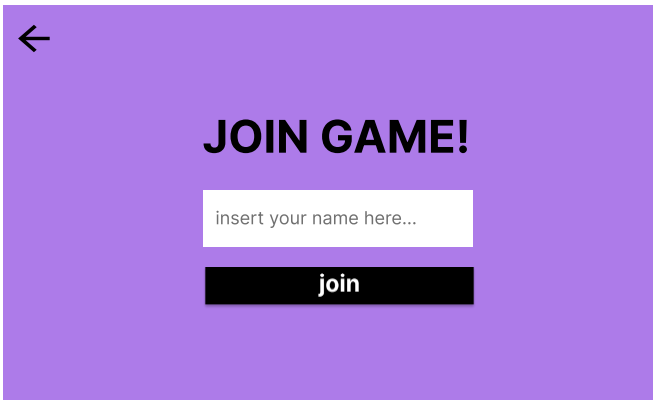


Start Game

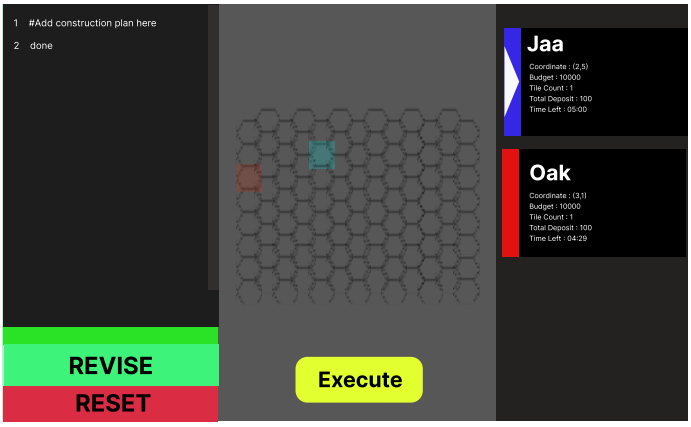
Game Setup



File Configuration



Join Game



Game Begin

Link: https://www.figma.com/file/2AUugpnPCK7OzltA4tmUwT/UI-Design?node-id=0%3A1&t=IP101U7CqOiFV1C4-1

Front-end

ในส่วนของ frontend จะใช้ Next.js Framework ในการพัฒนาหน้าเว็ป โดยจะใช้เครื่องมือในการพัฒนาหน้าเว็ป ได้แก่ HTML, CSS, React, และ JavaScript โดยทางฝั่ง frontend จะมีหน้าที่รับ input, render game (View and Controller), และส่งข้อมูลที่จำเป็นให้กับฝั่งของ Back-end ผ่าน RESTful API

Back-end

ในส่วนของ backend จะใช้ Spring Framework ในการสร้าง RESTful API สำหรับให้ฝั่ง Front-end เรียกใช้ โดยหน้าที่หลักๆของ backend คือ เก็บค่า state และ entities ต่าง ๆ ในตัวเกมเพื่อที่จะถูกนำไปแสดงผ่านทาง frontend และ ดำเนินการ operations ต่าง ๆ ภายในเกม (Model) ให้สมบูรณ์

Design Overview for UI

1.Start game page ในส่วนของหน้าจะมี ปุ่ม start ซึ่งเมื่อคลิกเข้าไปแล้วจะทำการเปลี่ยนไปยังหน้าของ game setup

2.Game setup page ในส่วนของหน้าจะมีปุ่มให้กดอยู่2ปุ่ม ได้แก่ ปุ่ม new game กับ ปุ่ม file config โดยหากกดปุ่ม new game ก็จะไปยังส่วนของหน้า Join game แต่หากกดปุ่ม file config ก็จะพาไปยังหน้าของ file configuration

3.File configuration page ในส่วนของหน้านี้ผู้เล่นสามารถที่จะแก้ไขไฟล์ได้ โดยหากกดปุ่ม done ก็ถือว่าการแก้ไขเสร็จสิ้น โดยการแก้ไขไฟล์นั้นก็จะส่งผลต่อตัวเกมโดยตรง

4.Join game page ในส่วนของหน้านี้ผู้เล่นจะทำการรอผู้เล่นฝั่งตรงข้าม ผู้เล่นสามารถที่จะเลือกรูปภาพแทนโปรไฟล์ของตัวเองได้และสามารถที่จะตั้งชื่อได้ เมื่อผู้เล่นอื่น ๆ เชื่อมต่อเข้ามาได้สำเร็จ ก็จะสามารถที่จะกดปุ่ม start ได้

5.Game start page ในหน้าเริ่มเกม ผู้สามารถที่จะซูมเข้าซูมออกบนแผนที่ได้ ในเกมมีการแสดงแถบเวลาและจำนวนรอบทั้งหมดบนตัวเกมและมีบาร์เปลี่ยน construction plan ที่ นอกจากนี้ที่บาร์ฝั่งขวาก็จะแสดงข้อมูลต่าง ๆ ของผู้เล่นทุกคน เช่น จำนวน budget เวลาที่เหลือในการทำ construction plan ฯลฯ โดยที่ เมื่อถึงรอบของผู้เล่นเวลาก็จะเริ่มนับถอยหลังลง

Main Components for UI

1.start game page : start button

2.game setup page : new game button, file config button, back button

3.file configuration page : done button, file configuration textbox

4.Join game page : start button, add name textbox, add image button, back button

5.Game page : construction plan bar, reset button, revise button, execute button, time count, territories, users information

แผนการทดสอบที่ 2

รายงานการทดสอบที่ 1

ในช่วงการพัฒนาจริง แต่ละคนในทีมจะแบ่งกันทดสอบส่วนของตัวเองด้วยการสร้างสถานการณ์สมมติบน parameter ทั้ง 3 ตัว ก่อนที่จะสร้าง *pull-request* เพื่อที่จะแก้ไขโค้ดหลัก แล้วค่อยทดสอบโค้ดรวมทีหลังอีกครั้ง ซึ่งกลับพบว่าทุกคนจะใช้วิธีการทดสอบทั้งแบบ *black-box testing* กับ *glass-box testing ร่วมกัน* ต่างจากที่วางแผนไว้ เนื่องจากเกิดความกังวลที่ว่าโปรแกรมจะสามารถแก้ไขได้ยากถ้าหากไปพบบัคทีหลัง

แผนการทดสอบที่ 2

จะทำการทดสอบโดยใช้หลักการของ *black-box testing* เป็นหลัก โดยจะสนใจที่ผลลัพท์ของส่งออกผ่านทางหน้าจอเป็นไปตาม construction plan ที่ผู้เล่นใส่เข้าไป ทั้งนี้การทดสอบรูปแบบนี้จะสามารถตรวจวัดที่ทั่งส่วน

ของ User Interface กับ ความครบถ้วนและความถูกต้องของ RESTful API ได้ โดย *input space* ที่ใช้ในการทดสอบควรครอบคุมกรณีผิดปกติด้วย เช่นการเกิด **`ArithmeticException`** หรือ *exceptions* อื่น ๆ ที่เกิดขึ้นในช่วงของ execution การที่ **`CityCrew`** เดินชนขอบเขต **`territory`** หรือการที่ผู้เล่นคนหนึ่งแพ้เกม เป็นต้น

แผนการทำงานที่ 2

แผนการทำงานที่ 2

การทำงานจะแบ่งออกเป็น 2 ช่วงเวลาดังนี้

ช่วงการทำงานที่	*รายละเอียดการทำงาน*	*วันเดดไลน์*
1	เชื่อม RESTful API กับ โปรแกรม parser and evaluator เดิม ด้วย Spring Framework และทำ components ของ UI ด้วย Next.js / React ให้เสร็จ	2/3/2023
2	เชื่อมโยงระหว่าง RESTful API เข้า front-end ให้เสร็จ	9/3/2023

โดยมีการแบ่งงานดังนี้

640610621	*640610639*
(ทำงานส่วนของ Front-end เป็นหลัก) – User Interface – เขียน components ด้วย Next.js ให้ครบ	(ทำงานส่วนของ Back-end เป็นหลัก) – Spring Framework – รองรับคำสั่งผ่านทาง RESTful API ให้ครบ (ซึ่งส่วนมากจะมีเพียงแค่ **GET** กับ **POST**) (Note: ในที่สุดจะเปลี่ยนไปใช้ *WebSocket*) – รองรับ *Dependency Injection* ของ Spring Framework
– เชื่อมโยง UI ให้เข้ากันกับ *game state* ผ่าน RESTful API	– ทดสอบความถูกต้องและดีบัคทั้งส่วนของทั้ง Back-end และ Front-end

ต่างจากคราวที่แล้ว การทำงานในครั้งนี้ แต่ละบุคคลในทีมจะทำงานที่ต่างกันมาก ซึ่งสามารถเป็นประโยชน์ได้จากที่การพัฒนาของทั้งสองฝั่งจะไม่ควบแน่นมากเกินจนไป ทำให้ตัวงานอาจสำเร็จได้เร็วขึ้น และมีเวลาสำหรับการแก้ไขและทดสอบส่วนของ parser and evaluator ได้ แต่ก็อาจส่งผลเสียให้ได้เช่นกัน จากที่ทั้ง Spring Framework กับ Next.js Framework เป็นเครื่องมือในการพัฒนาที่ทุกคนในทีมไม่ได้มีความถนัดสูงมาก

ปัญหาที่พบและสิ่งที่ได้เรียนรู้จากการพัฒนาที่ 2

สิ่งที่ได้เรียนรู้จากการพัฒนา *game state* กับ *User Interface*

- การใช้งานส่วนต่าง ๆ ของบน front-end จะขึ้นอยู่กับขั้นตอนการทำงานต่าง ๆ ของตัวเกม และทั้งขั้นตอนการทำงานของเกม กับ การทำงานของ front-end ก็ต้องขึ้นอยู่กับการสื่อสารระหว่างฝั่ง back-end กับ front-end ด้วย ซึ่งเห็นได้ว่าการพัฒนาในช่วงนี้มีการควบแน่นสูงมาก และต้องมีการพัฒนาในรูปแบบที่ค่อนข้างเป็นขั้นเป็นตอน ในช่วงการพัฒนานี้ก็ยังมีความสะดวกสบายบ้างอย่างที่ส่วนนี้จะไปพบบัคไม่บ่อยมาก หรือถ้ายังพบบัคก็จะไม่ได้แก้ไขได้ยากมากเท่าช่วงการพัฒนาส่วนของ parser กับ evaluator

ปัญหาที่พบเจอ

- ทั้งส่วนของ front-end กับส่วนของ back-end จะต้องพึ่งพาวิธีการสื่อสารระหว่างทั้งสองฝั่ง ซึ่งยังไม่ได้ทำออกมาให้ได้ชัดเจน ทำให้ผู้คนในทีมได้คำนึงว่าจะต้องมีการแก้ไขโค้ดเป็นจำนวนมาก
- ด้าน Next.js กับ Spring Framework เป็นส่วนที่ผู้คนในทีมยังไม่ได้มีประสบการณ์มาก ส่งผลให้คนในทีมต้องใช้เวลาในการศึกษาวิธีประยุกต์ใช้อุปกรณ์เหล่านี้ด้วย
- นอกจากมีเวลาในการทำงานมีไม่พอแล้ว มีงานการจากหน่วยวิชาอื่น ๆ อีก ยิ่งทำให้ผลงานไม่สมบูรณ์มากขึ้น มากไปกว่านั้น จากที่โปรเจคนี้ได้มาพัฒนามาค่อนข้างนานมากแล้ว เสริมกับที่ทุกคนในทีมมีหน้าที่อื่น ๆ นอกเหนือจากโปรเจค *UPBEAT* ทำให้ทุกคนเริ่มมีอาการหมดไฟ และยิ่งทำให้การพัฒนาล่าช้าไปมากกว่าที่ควรจะเป็นมาก

ส่วนของโปรเจคที่ยังไม่เสร็จสมบูรณ์

- ไม่ได้สร้างคลาส `Countdown` ที่จับเวลาผู้เล่น แต่สามารถคาดการได้แล้วว่าควรจะนำไปประกอบต่อเข้ากับฝั่งของ front-end ได้อย่างไร หรือจะเปลี่ยนการรองรับการนับถอยหลังไว้บน *front-end* เลย
- ค่า *configuration* ประเภทการนับถอยหลัง หรือค่า ที่ยังไม่ได้รองรับให้ส่งผลต่อการเล่นเกม
- RESTful API ไม่เพียงพอต่อการเล่นเกม ซึ่งแปลว่าจะต้องแทนที่ด้วย *WebSocket* ซึ่งมีการใช้ *publish–subscribe pattern* เข้าไปใช้งานด้วย (จากเดิมที่มีเพียงแค่ *request–response*)
- ตัวหน้าบอร์ด *Hexagonal Regions* ยังไม่เสร็จสมบูรณ์

server–client implementation

API endpoints and its listing specifications

Client's request	Request body	Server's publishment	Publishment body
/app/getConfig	–	/topic/config	Configurations `{ m:20,n:15,… }`
/app/setConfig	Configurations `{ m:20,n:15,… }`	/topic/cofig	
/app/join	Name of the player `"John Doe"`	/user/queue/token	Client's tokens and their crewId

			`{ token:“sample”, crewId:1 }`
/app/updateUsers	–	/topic/joinedUsers	Array of joined city crews `[CityCrew0, CityCrew1,…]`
/app/startGame	–	/topic/startGame	– (Front-end must changes its webpage once notified)
/app/setConstructionPlan	Client’s token, crewId, and revised construction plan `{ token:“sweetRelease”, crewId:1, constructionPlan:“done” }`	/user/queue/compileMessage	`{ isOkay: boolean, message: “String” }`
/app/execute	Client’s token, and crewId `{ token:“sweetRelease”, crewId:1, }`	/topic/alterations	The entire map `{ territory }`
/app/resign	Client’s token, and crewId `{ token:“sweetRelease”, crewId:1, }`	/user/queue/resignMessage	`{ isOkay: boolean, message: “D:” }`

แผนการทดสอบที่ 3

รายงานการทดสอบที่ 2

สำหรับการทดสอบรอบที่สองแล้ว ได้ใช้โปรแกรม Insomnia REST ในสั่งคำสั่งดำเนินการต่าง ๆ สำหรับ Game State โดย ตัวอย่างที่ใช้ในการทดสอบก็ควบคุมหลายกรณีในเกม เช่น กรณีผู้เล่นใช้ชื่อเดียวกัน กรณีมีผู้เล่นชนะ กรณีส่ง configuration ที่รับไม่ได้ กรณี Construction Plan มี Syntax error ซึ่งพบว่าการดีบัคจากการทดสอบช่วงการทำงานนี้ ไม่ได้มีความยุ่งยากเท่าช่วงออกแบบ parsers and evaluators ส่วนฝั่ง User interface จะมีทดสอบเพียงแค่ว่าหน้าเว็บต่าง ๆ สามารถเชื่อมเปลี่ยนไปหน้าเว็บอื่น ๆ ได้ถูกต้องหรือไม่

แผนการทดสอบที่ 3

การทดลองที่ 3 จะใช้ request *body* กับ *API endpoints* ที่เคยใช้ในการทดสอบที่ 2 มาปรับให้เหมาะสมกับสภาพแวดล้อมใหม่โดยเฉพาะที่ใช้ *WebSocket* ร่วมกับ/แทนที่ *HTTP request* รวมถึงการทดสอบ display ต่าง ๆ ที่ไม่สามารถแสดงได้ในการทดสอบรอบที่แล้ว เช่น การเชื่อมต่อของผู้เล่น หรือ แผนที่ *Hexagonal Region* โดยสรุป การทดสอบครั้งนี้จะใช้หลักการของ *black-box testing* เป็นหลัก โดยจะสนใจที่ผลลัพท์ของส่งออกผ่านทางหน้า User Interface

แผนการทำงานที่ 3

แผนการทำงานที่ 3

การทำงานจะแบ่งออกเป็น 2 ช่วงเวลาดังนี้

ช่วงการทำงานที่	*รายละเอียดการทำงาน*	*วันเดดไลน์*
1	เชื่อม WebSocket กับ โปรแกรม parser and evaluator เดิม ด้วย Spring Framework *สร้าง API endpoints* สำหรับการสื่อสารระหว่าง front-end กับ back-end และทำ components ของ UI ด้วย Next.js / Stomp ให้เสร็จ	1/4/2023
2	Putting in the finishing touches	3/4/2023

โดยมีการแบ่งงานดังนี้

640610621	*640610639*

(ทำงานส่วนของ Front-end เป็นหลัก)
- User Interface
  - เขียน components ด้วย Next.js ให้ครบ
  - เรียกใช้ *API endpoints* ต่าง ๆ บน front-end ให้ถูกต้องและเหมาะสม
	(ทำงานส่วนของ Back-end เป็นหลัก)
	- Spring Framework
	  - *สร้าง API endpoints* บน *back-end*

- Putting in the finishing touches	

(Forth Submission)

ปัญหาที่พบและสิ่งที่ได้เรียนรู้จากการพัฒนาที่ 3

สิ่งที่ได้เรียนรู้จากการพัฒนา *server-client implementation*

- ขณะช่วงการพัฒนาส่วน มีการเพิ่ม API endpoints นอกเหนือจากที่วางแผนไว้เยอะมาก จากเดิมที่มีเพียงแค่ 8 points บน 1 *controller class* ณ ปัจจุบันได้มีอยู่ทั้งหมด 18 endpoints บน 3 *controller classes* แสดงว่าโค้ดทั้งฝั่งของ back-end และ front-end ควรจะถูกออกแบบมาให้สามารถรองรับการต่อเติมใหม่ๆ ได้สะดวกมาก ซึ่งโชคดีพบว่าไม่ค่อยมีปัญหามากต่อโค้ดเดิมที่มีอยู่แล้ว เว้นแต่คลาส `CountdownClock` ที่เพิ่มมาใหม่จากการพัฒนารอบนี้ มีความเชื่อมโยงพึ่งพาต่อคลาสอื่น ๆ มาก ได้แก่ `CityCrew` `State` `Territory` และ *controller class* จากที่ต้องครอบคลุมหลายกรณีอย่าง การหยุดและการเริ่มทำงานของ *Thread* ให้ถูกต้องแล้วถูกเวลา การลบผู้เล่นออกจากเกมเมื่อหมดเวลา การเปลี่ยนใช้ระยะเวลาจาก *init_plan_time* เป็น *rev_plan_time* และการส่งข้อมูลการจับเวลาผ่าน WebSocket ด้วย

ปัญหาที่พบเจอ

- การจับเวลาผู้เล่นมีกรณีต่าง ๆ ให้ระมัดระวังอยู่มาก นอกจากการดีบัคมีความยากเทียบเท่าได้กับ *evaluators* ในช่วงการทำงานที่ 1 แล้ว การดีบัคส่วนนี้จำเป็นต้องพึ่งรอการจับเวลาให้หมดด้วย ทำให้ต้องใช้เวลากับส่วนนี้ไว้อยู่มาก
- การเชื่อมต่อระหว่าง front-end กับ back-end มีบัคอยู่เล็กน้อย เช่น ในบางกรณี ฝั่ง back-end ไม่ได้รู้เลยว่ามี client ที่ disconnect ออกจาก WebSocket กระทันหัน ส่งผลให้ back-end คิดว่ายังมีผู้เล่นคนนั้นยังอยู่ภายในเกม

- ฝั่ง front-end ยังคงเหลือส่วนที่ไม่เสร็จสมบูรณ์มากกว่าฝั่งของ back-end (ในขณะที่กำลังพัฒนา) เท่าให้สมาชิกที่เน่นการเขียนโปรแกรมฝั่ง back-end จำเป็นต้องไปช่วงฝั่งของ front-end อยู่บ้าง โดยเฉพาะส่วนของการเรียกใช้ API และการเชื่อมต่อระหว่าง client กับ back-end
- User Interface ของหน้า Game page ได้เปลี่ยนแปลงจากเดิมไว้มาก จากการที่เชื่อมต่อ API ได้แสดงให้เห็นว่าหน้า UI เดิมไม่สามารถต่อเติมข้อมูลผู้เล่น เมื่อมีผู้เล่นจำนวนมากด้วย นอกจากนี้ ก็ยังพบว่าการทำงานของเกมมี loop-holes อยู่มาก จึงออกแบบ UI ใหม่ให้ตอบสนองตามการทำงานของ loop-holes ไว้เลย โดยหวังผลว่าตัวเกิมจะได้เกิด *illusion* ที่ว่าผู้พัฒนาสามารถบังคับให้ผู้เล่นจำเป็นต้องเล่นตามที่ออกแบบไว้แต่แรกแล้ว

ส่วนของโปรเจคที่ยังไม่เสร็จสมบูรณ์

- จากแผนการทำงานของทีม โปรเจคนี้ทำมาได้ครบถ้วนสมบูรณ์แล้ว