ความเปลี่ยนแปลงที่เกิดขึ้นกับ Grammar

อย่างแรกโครงสร้างข้อมูลที่ได้ออกแบบใน Grammar ver เก่า นั้นถูกเปลี่ยนแปลงจากเดิมมากพอสมควรซึ่งจะบอกเป็นข้อๆ ไปดังนี้

1. Node ตอนแรกที่เราได้ออกแบบเป็น Interface เนื่องจากมี method ที่เราสามารถให้พูดคุยกันได้ แต่ว่าจากที่ได้ดูใน Grammar ทำให้ทราบว่า State กับ Expr มันต่อเนื่องแต่ไม่มีส่วนที่ต้องพูดคุยสื่อสารอะไรกัน ทำให้ต้องลบ method ที่ได้ออกแบบเอาไว้ใน Node ทิ้งไป ซึ่งทำให้ Interface ที่ว่างเปล่านั้นไม่มีอะไรให้ทำและดูเปล่าประโยชน์เกินไป จึงเปลี่ยนมันให้กลายเป็น class Node และให้ Node ย่อยที่เคยออกไปนั้นเป็น subclass ที่อยู่ใน Node ไปเลย ซึ่ง Node ที่ออกแบบไว้ตอนนี้มีอยู่ 2 Node คือ StateNode กับ ExprNode ซึ่งทั้ง 2 Node นี้จะทำงานในส่วนของพวกมันตามชื่อเลย
2. เนื่องจากการเปลี่ยนแปลงที่เกิดขึ้นในข้อ 1 ทำให้ทุกอย่างที่เคยใช้ Interface Node จะแก้เป็นไปใช้งาน Node.StateNode ไม่ก็ Node.ExprNode ตามที่เห็นใน Grammar ver ใหม่ ที่ส่งไปนะครับ
3. การเก็บข้อมูลของ ExprNode ในส่วนของ long eval(Map<String, Long> mem); ได้มีการเปลี่ยนแปลงภายในวงเล็บให้เป็น long eval(Game game); เนื่องจากมีความคิดใหม่ว่า การเก็บข้อมูลนั้นควรเอาไว้ในที่เดียวกัน และการที่ต้องทำ Game state ซึ่งเป็นส่วนที่น่าจะต้องเก็บข้อมูลเกือบทุกอย่างของเกมเอาไว้ทำให้เปลี่ยนที่เก็บข้อมูลจาก menไปเก็บข้อมูลใน game อย่างที่เห็นแทน ซึ่งในนั้นจะมี Map ของตัวมันอยู่
4. ซึ่งในส่วนของ StateNode ที่มี Node.StateNode evaluate(Game game) นั้น จะมีในส่วนของ AssignmentNode ที่ต้องเพิ่ม Node.StateNode evaluate(Map<String, Long> mem) เข้าไปด้วย เนื่องจากการเข้าถึงของทางฝั่งผู้เล่นที่จำเป็นต้องเก็บตัวแปรของตัวเองอย่างที่จารย์ชินเคยบอกเอาไว้ว่า ต่อให้ผู้เล่นทั้ง 2 ต่างก็ใช้ x เหมือนกัน แต่ก็ต้องไม่ไปปะปนกัน ดังนั้นในส่วนของ AssignmentNode จึงต้องเพิ่มเข้าไปอย่างที่บอกไปข้างต้นเพื่อให้เอา memory ของตัวเองเข้าไปบรรจุค่าตัวแปรที่เรา assign เข้าไป
5. ได้มีการสร้าง enum ที่ชื่อว่า Direction ขึ้นมาเพื่อเก็บทิศทางทั้ง 6 ของตัวเกมที่ต้องการเอาไว้ เพื่อให้เอาไปใช้กับ parseDirection ที่จะทำการอ่านทิศทางที่เราใส่ลงไปใน Construction plan และเปลี่ยนเป็นค่าที่เราสร้างเอาไว้ใน Direction ส่วนที่ว่าทำไมถึงสร้างขึ้นมานั้น เพื่อป้องกันความผิดพลาดที่อาจเกิดขึ้นระหว่างการเขียนโค้ด และทำให้เกิด HumanError แบบไม่ทันตั้งตัวได้
6. ได้มีการสร้าง List<String> Command และList<String> SpecialVariables ขึ้นมาใน Parser_im เพื่อให้ง่ายต่อการใช้งานและทำให้ตัว code ไม่ดูเปรอะจนเกินไป

7. จากในข้อ 6 List<String> SpecialVariables นั้นจะประกอบไปด้วยพวก command ต่างๆ อย่าง shoot, nearby, oppent และอื่นๆ ที่ถ้าผู้เล่นใช้งานคำพวกนี้เป็นตัวแปรจะแจ้งเตือนว่าห้ามใช้งาน และต้องกลับไปแก้ไข (ปล. สิ่งที่จารย์เขียนเอาไว้ใน ProjectSpec ก็ได้ใส่เอาไว้ในนี้แล้วด้วย)
8. แนวความคิดเกี่ยวกับ parseBlockStatement ตอนแรกคิดว่า BlockStatement นั้นจะเป็นในส่วนของตัวมันไปเลยอันอื่นจะไม่มีความเกี่ยวข้องกับ Block แต่หลังจากที่อ่านใน Construction plan อีกครั้ง ทำให้รู้ว่า Block นั้นสามารถใช้กับอะไรก็ได้ เหมือนกับการเขียนโค้ดปกติ ทำให้ BlockNode ที่ออกแบบเอาไว้ไม่ใช้งานอีกจึงหายไปจากระบบ แต่จำเป็นต้องเพิ่ม parse เข้ามาอีกอันหนึ่งชื่อว่า parseStatements เป็น parse ที่เอาไว้อ่านว่า State ต่อไปมีอยู่หรือไม่ ซึ่งมันจะวนอ่านไปจนกว่าจะไม่มี หรือว่าจะจบ BlockStatement
9. RegionNode ได้เปลี่ยนเป็น InvestNode กับ CollectNode แทน เพื่อให้ State ที่ได้นั้นต้องเป็นสิ่งที่เล็กที่สุดเท่าที่เป็นไปได้ตามหลักของ AST
10. ได้มีการสร้าง Error ขึ้นมาในแต่ละ Package ของ Grammar เพื่อให้สามารถอ่านได้อย่างสะดวกมากยิ่งขึ้นถ้าเราต้องการรู้ว่ามันจะเกิด Throw อะไรบ้างถ้าเราใช้สิ่งที่อยู่ใน Package นั้นๆ
11. ในส่วนของ Tokenizer ได้เพิ่มความสามารถที่อ่าน comment อย่างในตัวอย่างของ Construction plan ได้ และตัวแปรที่อีกฝ่ายเขียนมานั้นสามารถใส่ตัว _ ได้ เนื่องด้วยเหตุผลที่ว่าบางครั้งการเขียนติดกันอาจจะทำให้อ่านยากได้จึงคิดว่าการเพิ่มตัว _ นี้เข้าไปอาจจะทำให้อ่านตัวแปรแล้วรู้สึกโอเคมากขึ้น
12. ในส่วนของวิธีการอ่าน Construction plan นั้นได้เปลี่ยนไปใช้ File.readString() แทนการใช้ BufferedReader ที่เคยใช้มาเนื่องจาก readString นั้นจะเก็บข้อมูลที่เราอ่านได้จาก File มาเป็น String ตัวเดียว ส่วนถ้า File ที่มีหลายบรรทัดนั้น readString จะเก็บทุกอย่างอยู่ในบรรทัดเดียวแต่จุดที่ใน File ได้ขึ้นบรรทัดใหม่นั้นจะมีตัว \n ขั้นเอาไว้ ทำให้เวลาทดสอบ System.out.println() ออกมาจึงยังมีผลลัพธ์ไม่แตกต่างจาก File ที่เราไปอ่านมา ทำให้ในส่วนของ Tokenizer ได้มีการเพิ่มว่า ถ้าเจอ # หลังจากนั้นจะเป็นเนื้อหาของ comment จนกว่าจะเจอ \n ที่แปลว่าได้ขึ้นบรรทัดใหม่เป็นที่เรียบร้อยแล้ว
13. ซึ่งจากแนวคิดของข้อ 8 และข้อ 12 ที่ปูทางมาถึงขนาดนี้ทำให้ได้มีการเพิ่มบางอย่างเข้าไปใน StateNode ซึ่งก็คือ Node.StateNode nextState ถ้าอธิบายแบบง่ายๆ มันคือการที่ทุก Node จะมีการเชื่อมต่อไปยัง Node ถัดไป เพื่อให้ง่ายต่อการนำไปใช้งาน เพราะถ้า nextState เป็น null ก็จะสรุปได้ว่าการ parse นั้นได้จบลงไปแล้ว

14. จากข้อที่ 9 การเปลี่ยนแปลงที่เกิดขึ้นนั้นไม่ได้ส่งผลต่อโครงสร้างในส่วนของ Parser_im มีเพียงในส่วนที่ว่า InvestNode กับ CollectNode นั้นจะสนใจแต่ ExprNode เท่านั้น
15. จากการเปลี่ยนแปลงโครงสร้างหลักของ Node ในข้อ 1 และแก้ไขแนวคิดที่ออกแบบไว้คร่าวๆ ในครั้งแรก ทำให้การ evaluate ของ StateNode ต้องมีการ return ค่าที่เป็น StateNode เสมอ แทนที่ที่เคยเป็น void มาก่อน เพื่อให้นำไปใช้งานต่อได้ (การออกแบบในตอนแรกยังไม่ได้คิดถึงเรื่องการนำไปใช้งานต่อ)
16. มีการเปลี่ยนแปลงในส่วนของ WhileNode จากที่ว่าไป extends Node.StateNode เปลี่ยนไป extends IfNode แทน เนื่องจากโครงสร้างของ Node ทั้งสองนั้นมีความคล้ายคลึงที่มากพอสมควร จึงทำการ extends เพื่อนำ code มาใช้ซ้ำอย่างที่เคยเรียนมา ทำให้ในส่วนของ IfNode นั้นต้องเปลี่ยนจาก private เป็น protected ทั้งสามตัวอย่างที่เห็นใน Grammar ver ใหม่

What did your group learn from designing and implementing the parser and evaluator?

สิ่งที่ได้เรียนรู้จากการเขียน Grammar นั้นบอกเป็นข้อได้ดังต่อไปนี้

1. การออกแบบ เพราะสิ่งที่เราเคยออกแบบเอาไว้นั้นอาจจะไม่สามารถนำมาใช้ได้จริงทั้งหมดเพราะอาจจะยังไม่ได้คิดในเรื่องของการนำไปใช้งานต่อหรือลืมคิดเรื่อง บางอย่างไป ทำให้เวลาเขียนจริงมีการเปลี่ยนแปลงอย่างเลี่ยงไม่ได้
2. ได้ความรู้เรื่องใหม่ๆ มากพอสมควร อย่างการอ่านข้อมูลจาก File นั้นตอนแรกคิดจะใช้ BufferedReader อย่างที่เคยใช้ไปใน lab 09 ทว่าหลังจากที่ได้ไปลองถามจาก chatGPT และลองค้นหาข้อมูลใน Internet ทำให้ได้รู้ถึงทำสั่งอย่าง readString() ที่ทำให้เราไม่ต้องสนใจเรื่องการขึ้นบรรทัดใหม่ และต้องมาคิดถึงเรื่องของการอ่านบรรทัดใหม่อีก แค่ต้องไปเพิ่มให้ Tokrnizer_im ต้องอ่าน \n แล้วรู้ว่ามันคือการขึ้นรรทัดใหม่ จะแปลได้ว่าถ้าบรรทัดก่อนได้มีการเขียน comment เราจะเริ่มการเขียน command ใหม่ ไม่ใช่ comment นะ
3. การใช้งาน enum เป็นสิ่งที่ดีเพราะมันจะทำให้เราลดโอกาสเกิด HumanError ที่อาจเกิดจากการเขียน String ที่ผิดพลาดขึ้นมาได้ และทำให้เวลากลับมาอ่านโค้ดหรือทำการแก้ไขไม่ยุ่งยากจนเกินไป