COMP711 Software Assignment
Text-Based Adventure Game (Version 0)

Deliverables

1. An Antlr .g4 file containing your GameMap Grammar
2. An Antlr .g4 file containing your PlayerCommand Grammar
3. An archived file containing your IntelliJ Project with source code
4. Log of your software development activities as a PDF file

Supplied

1. An archive file of an IntelliJ project with starter code

Brief

Text-based adventure games have been around since the 1960s and have a simple interface in which the player types commands using the keyboard. These commands allow the player to move through and search the gaming *environment*, pick up objects and battle monsters.

For this assignment, you are to individually develop

1. Your own fully functional text-based adventure game engine based on the starting code described below
2. A **GameMap** grammar, using Antlr software to generate a parser for GameMap objects to be input from a text file into your text-based adventure game engine
3. A **PlayerCommand** grammar using Antlr software to generate a parser for player commands
4. A log of containing entries for software development activities you undertake for this project. Each log entry comprises a time stamp and a summary of progress. This can also be done using versioning software such as GitHub using commits

Gameplay

The player **explores** interconnected rooms picking up food, weapons and valuables to store in their inventory. The player wields weapons in their inventory to **battle** against monsters. The player wins if they find the exit and escape. The player loses the game if their health points reach 0.

COMP711 软件分配 T 基于 ext 的冒险游戏（版本 0）

交付

1. 包含游戏地图语法的 Antlr .g4 文件
2. 包含您的 PlayerCommand 语法的 Antlr .g4 文件
3. 包含带有源代码的 IntelliJ 项目的存档文件
4. 以 PDF 文件形式记录软件开发活动

提供

1. 带有起始代码的 IntelliJ 项目的存档文件

短

基于文本的冒险游戏自 1960 年代以来一直存在，并且具有一个简单的界面，玩家在其中使用键盘键入命令。这些命令允许玩家在游戏环境中移动和搜索，捡起物体并与怪物战斗。

对于此任务，您将单独开发

1. 您自己的基于文本的全功能冒险游戏引擎，基于下面描述的起始代码
2. 一种游戏地图语法，使用 Antlr 软件为游戏地图对象生成解析器，以便从文本文件输入到基于文本的冒险游戏引擎中
3. 使用Antlr软件为播放器命令生成解析器的PlayerCommand语法
4. 包含您为此项目进行的软件开发活动的条目的日志。每个日志条目都包含一个时间戳和一个进度摘要。这也可以使用版本控制软件（例如使用提交）的 GitHub 来完成。

游戏玩法

玩家探索相互连接的房间，捡起食物、武器和贵重物品以存储在他们的库存中。玩家在他们的库存中使用武器与怪物作战。如果玩家找到出口并逃脱，他们就会获胜。如果玩家的生命值达到 0，则玩家输掉游戏。

The player has health and confidence points which are boosted by eating food or admiring valuables in the inventory. Weapons are used to battle monsters and are stored in the inventory. Chests and other items may be found throughout the maze. Treasure chests contain valuables and are opened with a key and War chests contain weapons and are opened with a lockpick.

As the player moves between rooms, a monster may attack them. The player must **battle** the monster by wielding a weapon and exchanging attacks until one is defeated. During the battle, the player may wield any weapon in their inventory. The player's confidence and wielded weapon determine their attack strength. A monster's attacks decrease the health and confidence points of the player, based on the strength of the monster.

The tables below describe the commands available in **explore** and **battle** modes.

Explore Mode Commands	Description
	When the player enters a room, the following events occur: • A monster may randomly appear (and go to battle mode) • A description of the room is displayed
`door n`	Opens door labeled n and enter the room
`pickup item`	Pick up an item in room and add to inventory
`exit`	Search room to find exit.
`describe`	Describes the room, list pickups on the floor and number of doors available
`admire valuable`	Admire a valuable pickup in the inventory to increase confidence. The valuable may only be used once to increase confidence, but is not removed from the player's inventory.
`eat food`	Eats a food pickup in the inventory to increase health points. Once eaten, the food is removed from the player's inventory.
`stats`	Display player health and confidence points and inventory

`wield weapon`
`wield fistsoffury`
	Player wields weapon from inventory for battle
	Player wields fists of fury (does not appear in inventory)

`open chest`	Opens a treasure or war chest in the inventory. The contents of the chest is placed in the player's inventory and the chest removed.
`help`	Displays commands in this mode

玩家拥有生命值和信心点数，这些点数可以通过吃食物或欣赏物品栏中的贵重物品来提升。武器用于与怪物战斗并存储在库存中。箱子和其他物品遍布整个迷宫。宝箱包含贵重物品，用钥匙打开，战争箱包含武器，用锁撬打开。

当玩家在房间之间移动时，怪物可能会攻击他们。玩家必须通过挥舞武器和交换攻击来与怪物作战，直到一个被击败。在战斗中，玩家可以使用库存中的任何武器。玩家的信心和使用的武器决定了他们的攻击力。怪物的攻击会根据怪物的强度降低玩家的生命值和信心点数。

下表描述了探索和战斗模式中可用的命令。

探索模式命令	描述
	当玩家进入房间时，会发生以下事件： • 怪物可能会随机出现（并进入战斗模式） • 显示房间的描述
门 n	打开标有 n 的门并进入房间
提货物品	在房间中拾取物品并添加到库存出口 搜索房间以查找出口。
描述	描述房间，列出地板上的拾取器和门数 可用
欣赏有价值的	欣赏库存中的贵重拾取以增加信心。贵重物品只能使用一次以增加信心，但不会从玩家的物品栏中移除。
吃食物	吃物品栏中的食物拾取以增加生命值。一次 吃掉后，食物会从玩家的物品栏中移除。
统计数据	显示玩家生命值和置信点以及库存
挥舞武器 挥舞拳头	玩家挥舞着物品栏中的武器进行战斗 玩家挥舞着愤怒的 拳头（未出现在物品栏中）
打开宝箱	打开物品栏中的宝藏或战争宝箱。的内容 宝箱被放置在玩家的物品栏中，宝箱被移除。
帮助	在此模式下显示命令

Battle Mode Commands	Description
`attack`	Attacks the monster in the room using the wielded weapon

`wield weapon`
`wield fistsoffury`
	Player wields weapon from inventory for battle
	Player wields fists of fury (does not appear in inventory)

`help`	Displays commands in this mode

Grammar Designs

You will design and implement two simple grammars for this assignment.

GameMap Grammar

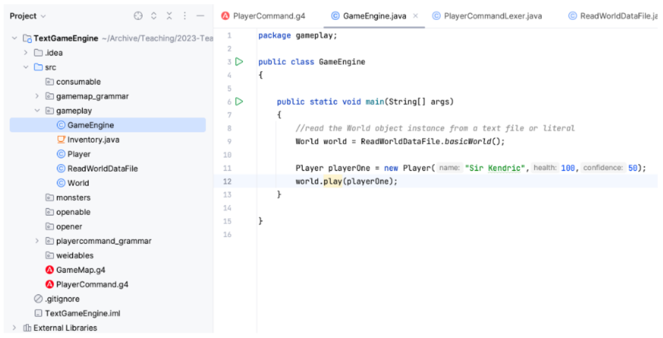
The gaming environment comprises interconnected rooms. The player begins the game in a room designated as the start and finishes the game in an exit room. Design the GameMap grammar with syntactic structures corresponding to the graph structure of the gaming environment, including rules to parse and build Room objects; according to the class description in the next sections.

PlayerCommand Grammar

The player interacts with the gaming environment according to the explore and battle mode commands. Design the PlayerCommand grammar with syntactic structures corresponding to the structure of the commands in explore and battle mode.

Getting Started

You have been supplied with an archive file **TextAdventureGame.zip** to import into your IntelliJ workspace. In IntelliJ, go to File -> Open. Click the archive file and then click Finish. The TextGameEngine Project will look similar to:



```
TextGameEngine ~/Archive/Teaching/2023-Tea
  .idea
  src
    consumable
    gamemap_grammar
    gameplay
      GameEngine
      Inventory.java
      Player
      ReadWorldDataFile
      World
    monsters
    openable
    opener
    playercommand_grammar
    weidables
    GameMap.g4
    PlayerCommand.g4
  .gitignore
  TextGameEngine.iml
External Libraries
```

```java
package gameplay;

public class GameEngine
{

    public static void main(String[] args)
    {
        //read the World object instance from a text file or literal
        World world = ReadWorldDataFile.basicWorld();

        Player playerOne = new Player( name: "Sir Kendric", health: 100, confidence: 50);
        world.play(playerOne);
    }

}
```

战斗模式命令	描述
攻击	使用挥舞的武器攻击房间内的怪物
挥舞武器	玩家挥舞着物品栏中的武器进行战斗
挥舞拳头	玩家挥舞着愤怒的拳头（未出现在物品栏中）
帮助	在此模式下显示命令

语法设计

您将为此作业设计和实现两个简单的语法。

游戏地图语法

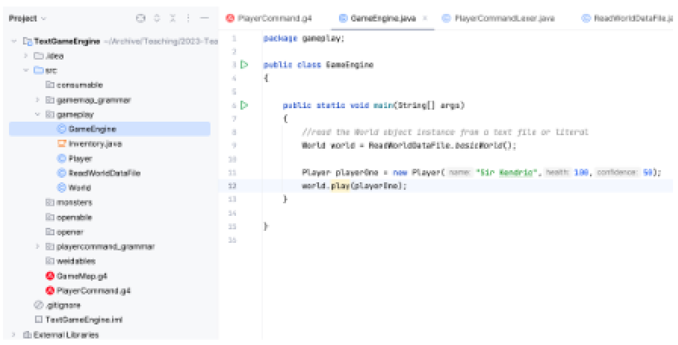
游戏环境包括连通房。玩家在指定为开始的房间开始游戏，并在退出房间结束游戏。使用与游戏环境的图形结构相对应的语法结构设计 GameMap 语法，包括解析和构建 Room 对象的规则;根据下一节中的类描述。

玩家命令语法

玩家根据探索和战斗模式命令与游戏环境进行交互。使用与探索和战斗模式中的命令结构相对应的语法结构设计 PlayerCommand 语法。

开始

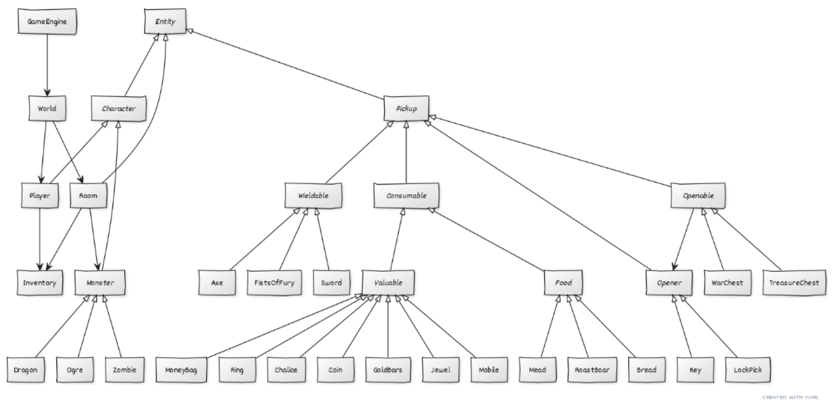
已为您提供一个存档文件 TextAdventureGame.zip以导入到您的 IntelliJ 工作区中。在 IntelliJ 中，转到“文件”->“打开”。单击存档文件，然后单击“完成”。TextGameEngine 项目将类似于：



```java
package gameplay;

public class GameEngine
{

    public static void main(String[] args)
    {
        //read the World object instance from a text file or literal
        World world = ReadWorldDataFile.basicWorld();

        Player playerOne = new Player( name: "Sir Kendric", health: 100, confidence: 50);
        world.play(playerOne);
    }

}
```

Class Hierarchy

The GameEngine class hierarchy comprises many classes. However, the actual coding for each class is very light.



Entity Class Hierarchy

Implement the *Entity* class hierarchy shown in the Class Hierarchy Diagram. Start by defining the abstract class *Entity*. Abstract classes are denoted are italicised in the diagram. All entities in the game will need a description instance variable and a unique identifier instance variable id. Create the Entity(String) constructor which takes a description and sets the unique identifier as

```
this.id = this.getClass().getSimpleName();
```

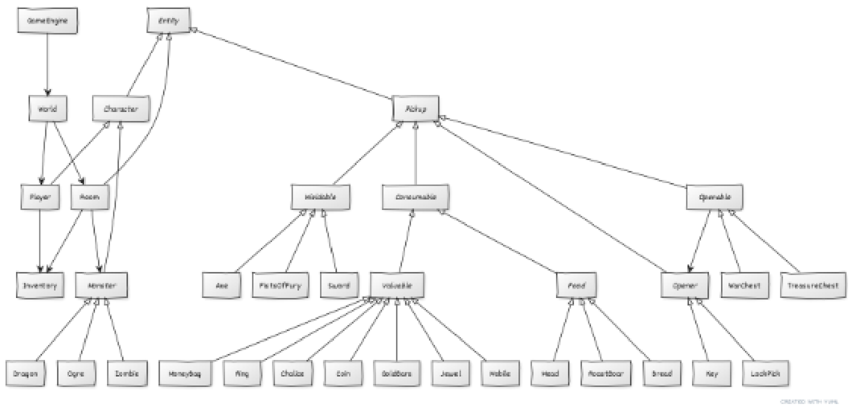
Add get/set methods for the description. Assume throughout that we will need get and set methods for most instance variables unless stated otherwise. Add a get method for **id**.

The entities in our game will often exhibit randomized behaviour. Write a protected method which returns a random integer between two numbers x and y (exclusive):

```
return new Random().nextInt(y-x) + x;
```

类层次结构

游戏引擎类层次结构由许多类组成。但是，每个类的实际编码非常轻。



实体类层次结构

实现类层次结构图中所示的实体类层次结构。首先定义抽象类实体。抽象类在图中表示为斜体。游戏中的所有实体都需要一个描述实例变量和一个唯一标识符实例变量 id。创建实体（字符串）构造函数，该构造函数采用描述并将唯一标识符设置为

```
this.id = this.getClass（）.getSimpleName（）；
```

为说明添加获取/设置方法。假设我们整个实例变量都需要 get 和 set 方法，除非另有说明。为 id 添加一个 get 方法。

我们游戏中的实体通常会表现出随机行为。编写一个受保护的方法，该方法返回两个数字 x 和 y（不包括）之间的随机整数：

```
返回新的 Random（）.nextInt（y-x） + x;
```

Next, write the compareID(String) method, which determines the entity's id equality to a given string, ignoring the case. Lastly, the toString method should return the entity's id.

Pickup Class Hierarchy

Pickup is an abstract class. It contains a constructor to initialize the description.
The classes *Opener*, *Openable*, *Consumable* and *Wieldable* extend pickup:
Lockpick and Key classes extend *Opener* and have constructors to initialize their descriptions.

The *Openable* class has an instance variable for boolean locked and also has a *Pickup* contents instance variable. The constructor should set these values.
Consumable maintains a boolean instance variable to determine if the object has been consumed.

Wieldable Class Hierarchy

Wieldable is an abstract class which has two integer value instance variables high and low. The **hit** method returns a random number x such that lo <= x < high. A constructor should initialize all instance variables. There are three subclasses; refer to the class diagram and implement each one, defining the necessary constructor for each class.

Consumable Class Hierarchy

Consumable is an abstract class and *Valuables* is an abstract class extending *Consumable* that maintains a number representing the object's value. There are seven subclasses; refer to the class diagram and implement each one, defining an appropriate constructor for each. No new methods or data are needed for these subclasses.

Food is an abstract class which stores a number representing health points. Refer to the class diagram for the names of the three subclasses extending Food. They do not require any additional methods or data.

Character Class Hierarchy and *Monster* and *Player* subclasses

The *Character* abstract class has an instance variable to keep track of health points which should always store a non-negative number and a constructor to instantiate all instance variables. *Character* contains abstract methods

- `protected abstract int dealAttackDamage();`
- `public abstract int defendAttack(Character enemy);`

接下来，编写 compareID（String） 方法，该方法确定实体的 id 与给定字符串的相等性，忽略大小写。最后，toString 方法应返回实体的 id。

拾取类层次结构

拾取是一个抽象类。它包含一个用于初始化说明的构造函数。类打开器、可打开、消耗品和可使用扩展拾取：锁撬和钥匙类扩展打开器，并具有构造函数来初始化其描述。

Openable 类具有用于布尔锁定的实例变量，并且还具有 PickContent 实例变量。构造函数应设置这些值。

消耗品维护一个布尔实例变量，以确定对象是否已被使用。

可使用类层次结构

Wieldable 是一个抽象类，它有两个整数值实例变量 high 和 low。hit 方法返回一个随机数 x，使得 lo <= x <高。构造函数应初始化所有实例变量。有三个子类;参考类图并实现每个类图，为每个类定义必要的构造函数。

易耗品类层次结构

消耗品是一个抽象类，Valuable s 是一个抽象类，扩展了易耗品，它维护一个表示对象值的数字。有七个子类;参考类图并实现每个类图，为每个类图定义适当的构造函数。这些子类不需要新的方法或数据。

食物是一个抽象类，它存储了一个代表健康点的数字。有关扩展 Food 的三个子类的名称，请参阅类图。它们不需要任何其他方法或数据。

角色类层次结构以及怪物和玩家子类

Character 抽象类有一个实例变量来跟踪运行状况点，该变量应始终存储一个非负数和一个构造函数来实例化所有实例变量。字符包含抽象方法

- 受保护的抽象int dealAttackDamage（）；
- 公共抽象防御攻击（角色敌人）；

The *Monster* subclass of *Character* and has integer values storing the *probability* of the monster appearing in a room and the amount of *damage* the Monster can deal in an attack. The constructor initializes these values and the description.

The *Monster* subclass overrides the dealAttackDamage and defendAttack methods. The dealAttackDamage method returns the value *damage*+r, where r is a randomly selected value between 1 and 10. The defendAttack method simulates an incoming attack from an enemy character. The method invokes d = enemy.dealAttackDamage() and reduces the character's health by d. The value d is returned by the method to report how much damage was dealt to the character.

The *Monster* subclass has a method boolean appear() to determine whether or not the monster appears in a room. If the monster's health is 0, false is returned. Otherwise, the monster appears with frequency based on its *probability* value. A simple way to implement this method would be to pick a random number x between 0 and 101 (exclusive) and return x<=*probability.*

The *Player* class has instance variables for name, health and confidence points, a Wieldable weapon, and an Inventory. You can add any other instance variables needed.
The *Player* class overrides the following methods:

- int dealAttackDamage(), calculates and returns the damage **d** dealt by the player's attack according to the formula **d** := **h** + **h*****c**/100, where **h** is the strength of the user's weapon and **c** is the confidence of the player
- int defendAttack(Character enemy), calculates and returns the damage **d** from the enemy's **d** = enemy.dealAttackDamage and decreases the player's health by **d**. Additionally, the player's confidence is decreased by **d**/2.

Room class

The Room class contains instance variables to store if the room is the final room (and has an exit so the play can finish the game), and a primitive array **Room**[] of all rooms connected to the room. The Room objects maintains a list of Pickup objects present in the room by way of the **Inventory** class, which maintains an ArrayList. ArrayList is a dynamic array, in which elements can be added and removed. The Room object maintains an instance variable of the monster (or monsters) residing in the room. The Room class will need the default constructor (which initializes instance variables to null values) and a constructor with signature

```
public Room(String description,Inventory pickupsInRoom,
            Room[] connectingRooms)
```

Character 的怪物子类，具有整数值，用于存储怪物出现在房间中的概率以及怪物在攻击中可以造成的伤害量。构造函数初始化这些值和说明。

怪物子类覆盖了 dealAttackDamage 和 defendAttack 方法。方法返回值 damage+r，其中 r 是随机选择的值，介于 1 和 10 之间。防御攻击方法模拟来自敌方角色的攻击。该方法调用 d = enemy.dealAttackDamage（） 并将角色的生命值降低 d。该方法返回值 d，以报告对字符造成的伤害。

Monster 子类有一个方法布尔 *appear*（） 来确定怪物是否出现在房间中。如果怪物的生命值为 0，则返回 *false*。否则，怪物会根据其概率值出现频率。实现此方法的一种简单方法是在 0 到 101（不包括）之间选择一个随机数 x，并返回 x<=概率。

Player 类具有名称、生命值和置信点、可使用武器和物品栏的实例变量。您可以添加所需的任何其他实例变量。Player 类重写以下方法：

- int dealAttackDamage（），根据公式 d ： = h + h*c/100 计算并返回玩家攻击造成的伤害 d，其中 h 是用户武器的强度，c 是玩家的信心

- int defendAttack（角色敌人），计算并返回敌人的伤害d=敌人.dealAttackDamage，并降低玩家的生命值d。此外，玩家的信心会降低 d/2。

房间类

Room 类包含实例变量，用于存储房间是否是最后一个房间（并且有一个出口，以便游戏可以完成游戏），以及连接到房间的所有房间的基元数组 Room[]。Room 对象通过 Inventory 类维护房间中存在的拾取对象列表，该类维护 ArrayList。ArrayList 是一个动态数组，可以在其中添加和删除元素。Room 对象维护驻留在房间中的怪物（或多个怪物）的实例变量。Room 类将需要默认构造函数（将实例变量初始化为 null 值）和带签名的构造函数

公共房间（字符串描述、库存取货在房间、房间[]连接房间）

The World Class

The World class implements the core playable functionality of the text game in the play(Player) method. This method is essentially a while loop to parse and process input from the player depending on the current mode: battle or explore. The while loop continues until the game halts: when the player wins, loses or quits the game.

You will need to write the code for private methods

`processExploreUserInput()` and `processBattleUserInput()`

which prompts the player for input using a **Scanner** and processes it as defined by your PlayerCommand Antlr grammar. Your code should be able to handle errors such as mistyped commands.

Processing explore-mode commands

Suppose processExploreUserInput reads the text *"pickup JEWEL"* from the player, storing the player's typed input in the variable String input. Use the generated Antlr code to parse this string. You can ask the Lexer object to get the next token from the input; e.g.

Token token = lexer.nextToken();

Should be compared to a list of commands possible in the explore mode, using a switch statement. Since the player has invoke the pickup command, perhaps a private pickup() method in the World class is invoked. This method would determine if a *Pickup* object with an id "jewel" was present in the room's inventory (pickupsInRoom). If so, it would be removed from the room and placed in the player's inventory.

As another example, suppose the player later types admire goldbars in the explore mode prompt and the admire() method is invoked, which carries out the following steps:

1. Determine if goldbars is in the player's inventory (use Inventory's select method)
2. If so, then `Pickup pickup = player.getInventory().select(id)`
3. Use the code `(pickup instanceof Valuable)` which returns true if pickup refers to a Valuable object.
4. Since GoldBars is indeed a subclass of Valuable, we cast `Valuable valuable = (Valuable) pickup;`
5. Then, we are able to invoke `this.player.admire(valuable);`

All other commands are structured similarly.

世界级

World 类在 play（Player） 方法中实现了文本游戏的核心可玩功能。这种方法本质上是一个while循环，用于根据当前模式解析和处理来自玩家的输入：战斗或探索。while 循环一直持续到游戏停止：当玩家赢、输或退出游戏时。

您需要为私有方法编写代码

processExploreUserInput（） 和 processBattleUserInput（）

它使用扫描仪提示玩家输入，并按照您的玩家命令 Antlr 语法的定义对其进行处理。代码应该能够处理错误，例如键入错误的命令。

处理探索模式命令

假设 processExploreUserInput 从播放器读取文本“拾取 JEWEL”，将玩家键入的输入存储在变量 String 输入中。使用生成的 Antlr 代码解析此字符串。您可以要求词法分析器对象从输入中获取下一个令牌;例如

Token token = lexer.nextToken（）;

应使用 switch 语句与探索模式下可能的命令列表进行比较。由于玩家已经调用了 picking命令，因此可能会调用 World 类中的私有 pickup（） 方法。此方法将确定房间的库存中是否存在 ID 为“宝石”的拾取对象 （pickupsInRoom）。如果是这样，它将被从房间中移除并放置在玩家的物品栏中。

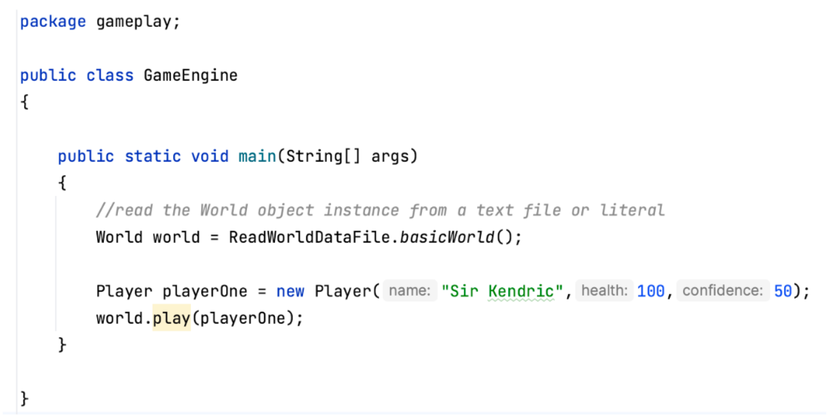
再举一个例子，假设玩家稍后在探索模式提示符中键入欣赏金条，并调用 admire（） 方法，该方法执行以下步骤：

1. 确定金条是否在玩家的物品栏中（使用物品栏的选择方法）
2. 如果是这样，则拾取 = player.getInventory（）.select（id）
3. 使用代码（有价值的拾取实例），如果拾取引用有价值的对象，则返回 true。
4. 由于金条确实是有价值的子类，我们铸造有价值的价值=（有价值的）拾取;
5. 然后，我们可以调用这个.player.admire（有价值的）;

所有其他命令的结构类似。

The GameEngine Class

The GameEngine class contains a main method which reads a World object from a static ReadWorldDataFile class method. These methods use Antlr to parse Room objects using the GameMap grammar.



```
package gameplay;

public class GameEngine
{

    public static void main(String[] args)
    {
        //read the World object instance from a text file or literal
        World world = ReadWorldDataFile.basicWorld();

        Player playerOne = new Player( name: "Sir Kendric", health: 100, confidence: 50);
        world.play(playerOne);
    }

}
```

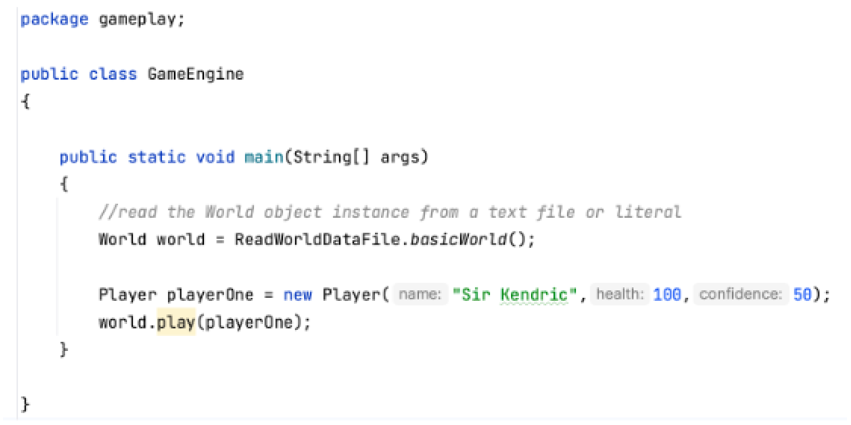
Possible extensions to the game

According to the marking rubric, to receive a grade in the A range, you will need to extend the gameplay beyond the assignment instructions. Use your imagination! Some possible suggestions:

- Commands to combine items in your inventory
- Extended Battle commands to de-escalate battles
- New and interesting items
- Teleportation, randomisation of the Player's position in the maze
- Your idea here

游戏引擎类

类包含一个 main 方法，该方法从静态 ReadWorldDataFile 类方法读取 World 对象。这些方法使用 Antlr 使用游戏地图语法解析房间对象。



```java
package gameplay;

public class GameEngine
{

    public static void main(String[] args)
    {
        //read the World object instance from a text file or literal
        World world = ReadWorldDataFile.basicWorld();

        Player playerOne = new Player( name: "Sir Kendric", health: 100, confidence: 50);
        world.play(playerOne);
    }

}
```

游戏的可能扩展

根据评分标准，要获得 A 范围内的成绩，您需要将游戏玩法扩展到作业说明之外。发挥你的想象力！一些可能的建议：

- 用于组合库存中物品的命令 - 扩展战斗命令以降级战斗 - 新的和有趣的物品
- 瞬移，随机化玩家在迷宫中的位置 - 你的想法在这里

Marking Rubric

Criteria:	Weight:	Grade A Range 100 ≥ x ≥ 80%	Grade B Range 80 > x ≥ 65%	Grade C Range 65 > x ≥ 50%	Grade D Range 50 > x ≥ 0%
GameMap Grammar Implementation	**50%**	Concise syntactic structures are used to define the gaming environment. Map graph structure and room contents/ properties can be specified by an input file. Interesting extensions to the grammar are created.	Concise syntactic structures are used to define the gaming environment. Map graph structure and room contents/ properties can be specified by an input file.	Problems in reading game environments but mostly works.	Absent functionality for parsing World data or code does not compile
Entity Class Hierarchy Implementation	**20%**	All classes present with appropriate structure and method overriding	Some minor issues with hierarchy, constructors	Code not appropriately overridden in classes	Absent classes or functionality or code does not compile
PlayerCommand Grammar and Game Play	**30%**	Excellent handling of text and command entries using the PlayerCommand grammar. Data structures are consistently maintained during gameplay. Interesting extensions of the grammar are created.	Excellent handling of text and command entries using the PlayerCommand grammar. Data structures are consistently maintained during gameplay.	Problems handling commands in either gameplay modes	Absent functionality for parsing text or commands or code does not compile

Javadoc Commenting

Please comment your code with your name

```
/**
  * Comment describing the class.
  * @author yourname studentnumber
 **/
```

Submission Instructions

Submit an archive of all required deliverables to Canvas before the due date.

评分量规

标准：	重量：	A级范围 100 ≥ x ≥ 80%	B级范围 80 > x ≥ 65%	C 级范围 65 > x ≥ 50%	D级范围 50 > x ≥
游戏地图语法实现	**50%**	简洁的语法结构用于定义游戏环境。地图结构和房间内容/属性可以通过输入文件指定。有趣的扩展	简洁句法结构用于定义游戏环境。地图结构和房间	阅读游戏环境中的问题，但大部分有效。	缺少解析 World 数据或代码的功能无法编译
实体类层次结构实现	**20%**	所有类都存在适当的结构和方法重写	层次结构、构造函数的一些小问题	代码不是适当地在类中被覆盖	缺课或功能或代码无法编译
玩家命令语法和游戏玩法	**30%**	使用 PlayerCommand 语法出色地处理文本和命令条目。在游戏过程中始终保持数据结构。创建了语法的有趣扩展。	使用玩家命令语法。在游戏过程中始终保持数据结构。	在任一游戏模式下处理命令时出现问题	缺少用于解析文本或命令或代码无法编译的功能

Javadoc 注释

请用您的名字注释您的代码

```
        /**
          *  描述课程的评论。  * @author您的姓名学
          生编号  **/
```

投稿说明

在截止日期之前将所有必需的可交付成果的存档提交给 Canvas。