The Design and Development of a Dungeons and Dragons Game Library and an Example Client Interface

Calvin Withun and CJ Chung

College of Arts & Sciences, Lawrence Technological University

INTRODUCTION

Dungeons and Dragons is a classic tabletop role-playing game which was first publicized in 1974. This game has remained popular over the years, and continues to get new additions to the current day. However, the ongoing Covid-19 pandemic has made it challenging for players to engage with the game. Dungeons and Dragons was designed to be played around a table with a group of other people, which is a problematic environment for those who are attempting to maintain social distancing. There are, of course, certain online resources such as www.roll20.com or www.dndbeyond.com which provide a virtual platform for playing the game, and these resource have been available since long before the outbreak began. While these resources typically attempt to provide some degree of automation for common events in the game, such as making attack rolls and saving throws, there is a great deal of game content which still requires user maintenance. The objective of this project is to create a game library which eliminates this user maintenance by automating all interactions between core game elements. This automation will serve many purposes, such as allowing newer players to engage with the game without an in-depth knowledge of the game mechanics, allowing players to focus on the role-playing aspect of the game without worrying about the math governing character interactions, or opening the door to certain game mechanics which typically bog down gameplay due to their scale or complex nature.

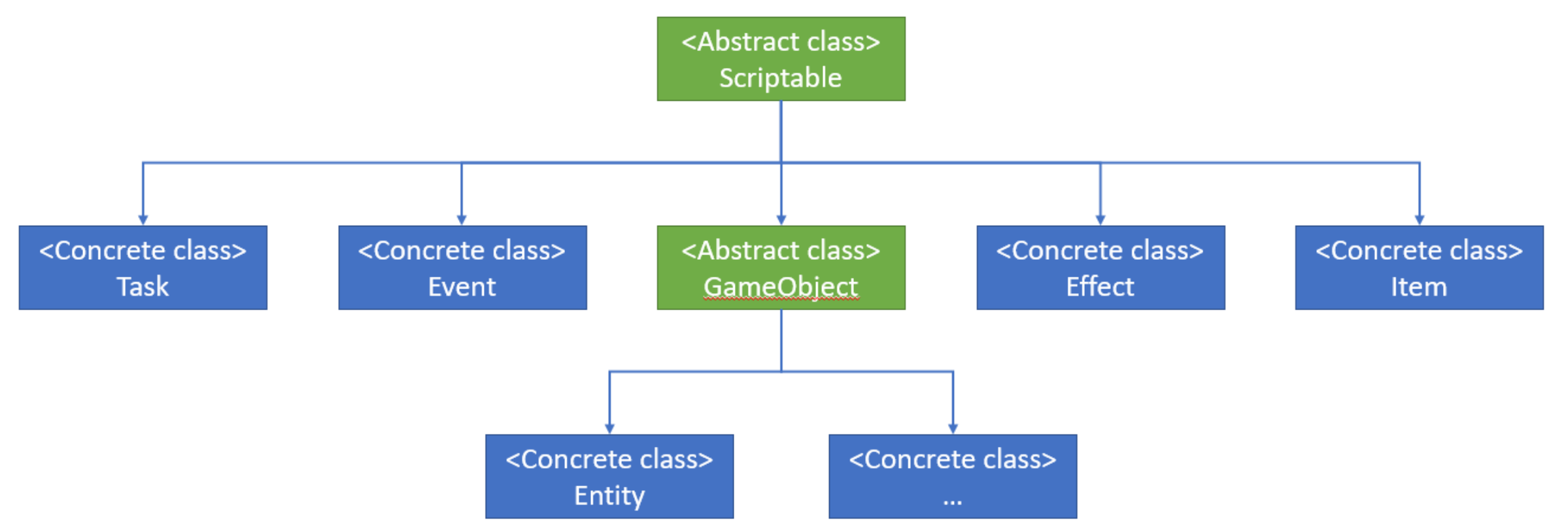


Figure 1: Core Data Types Class Diagram
The "…" class block represents yet-unimplemented concrete derivations of GameObject

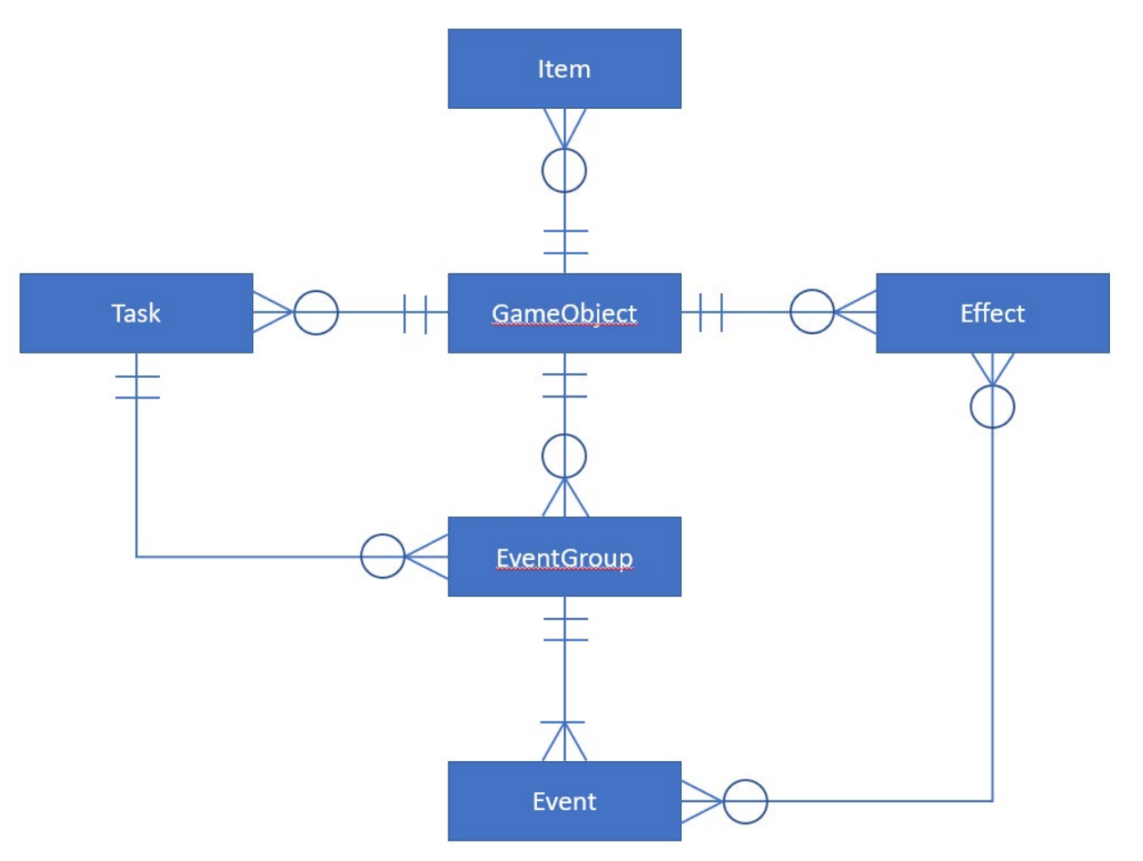


Figure 2: Entity Relationship Diagram for Core Data Types

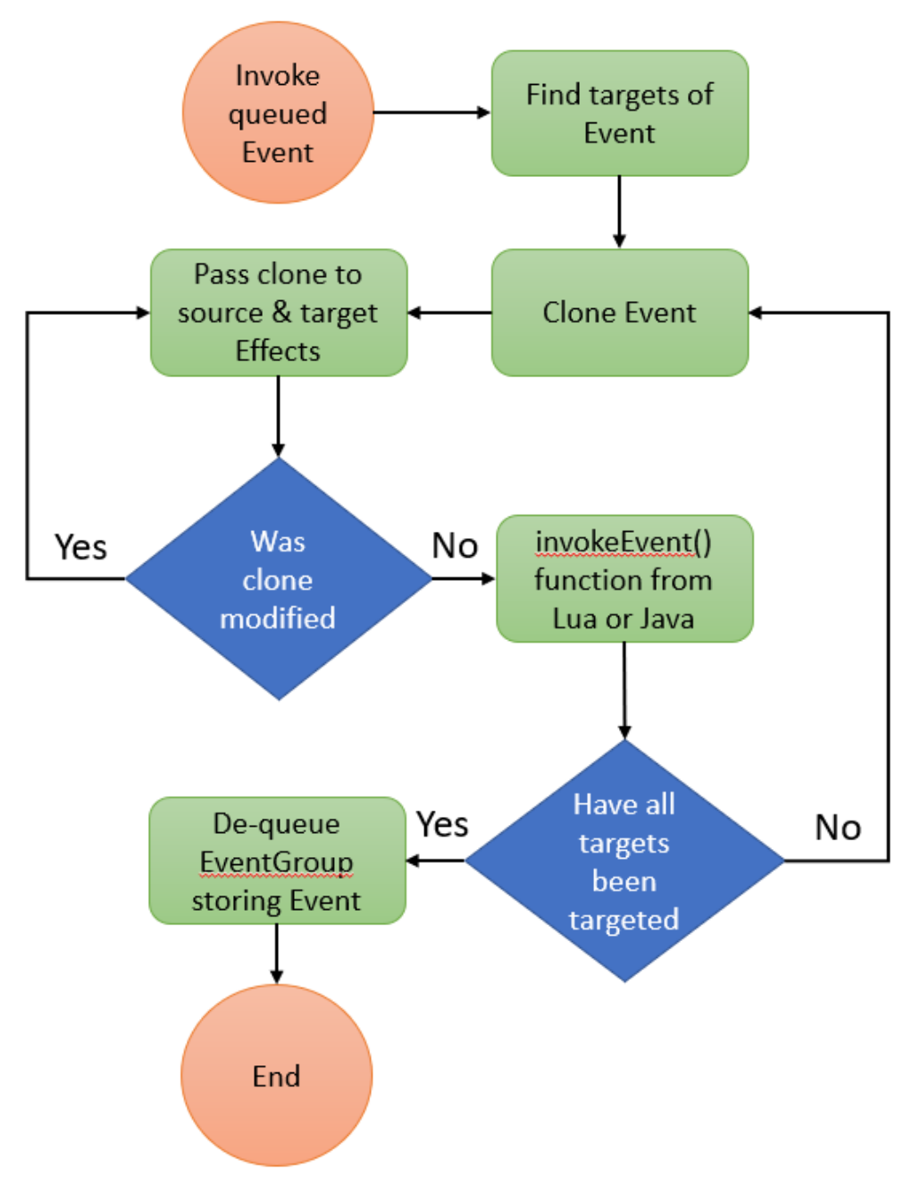

Invoke queued Event
Find targets of Event
Clone Event
Pass clone to source & target Effects
Was clone modified
Yes
No
invokeEvent() function from Lua or Java
Have all targets been targeted
No
Yes
De-queue EventGroup storing Event
End

Figure 3: Flow Diagram for Invoking Queued Events

METHODS

Abstracting Features of the Game

Dungeons and Dragons is a game which is regulated by a complex system of rules. However, any given rule is typically straightforward and unambiguous. Furthermore, all game rules appear to adhere to certain patterns. As it is discussed in *Results & Summary*, this pattern was eventually discovered to be applicable to most games rather than exclusively to Dungeons and Dragons. However, the existence of this pattern allows for the game features to be abstracted into a smaller number of developer-friendly data types. There are currently five core elements identified in this pattern: game pieces, potential behavior of game pieces, actualized behavior of game pieces, modifiers applied to game pieces, and items or artifacts which can be used or controlled by game pieces. These elements are represented by Java data types as follows:

1. GameObject: this data type represents any game piece which would appear on the game board.
2. Task: this data type represents a set of potential behavior available to a GameObject, as well as the resource cost for actualizing the behavior.
3. Event: this data type represents GameObject behavior which has been actualized in virtue of invoking a Task.
4. Effect: this data type represents a modifier which has been applied to a GameObject.
5. Item: this data type represents any physical artifact which might be controlled or possessed by a GameObject.

The library includes a small selection of default derivations of these data types for common Dungeons and Dragons phenomena, such as making attack rolls and saving throws or having proficiency with certain items, but these data types are designed to reference Lua scripts external to the library. These Lua scripts define the specific behavior of these core objects, allowing the client software to introduce customized game content simply by providing a collection of Lua scripts. The library then facilitates the interactions between these customized data types. The process of invoking an Event is detailed in Figure 3.

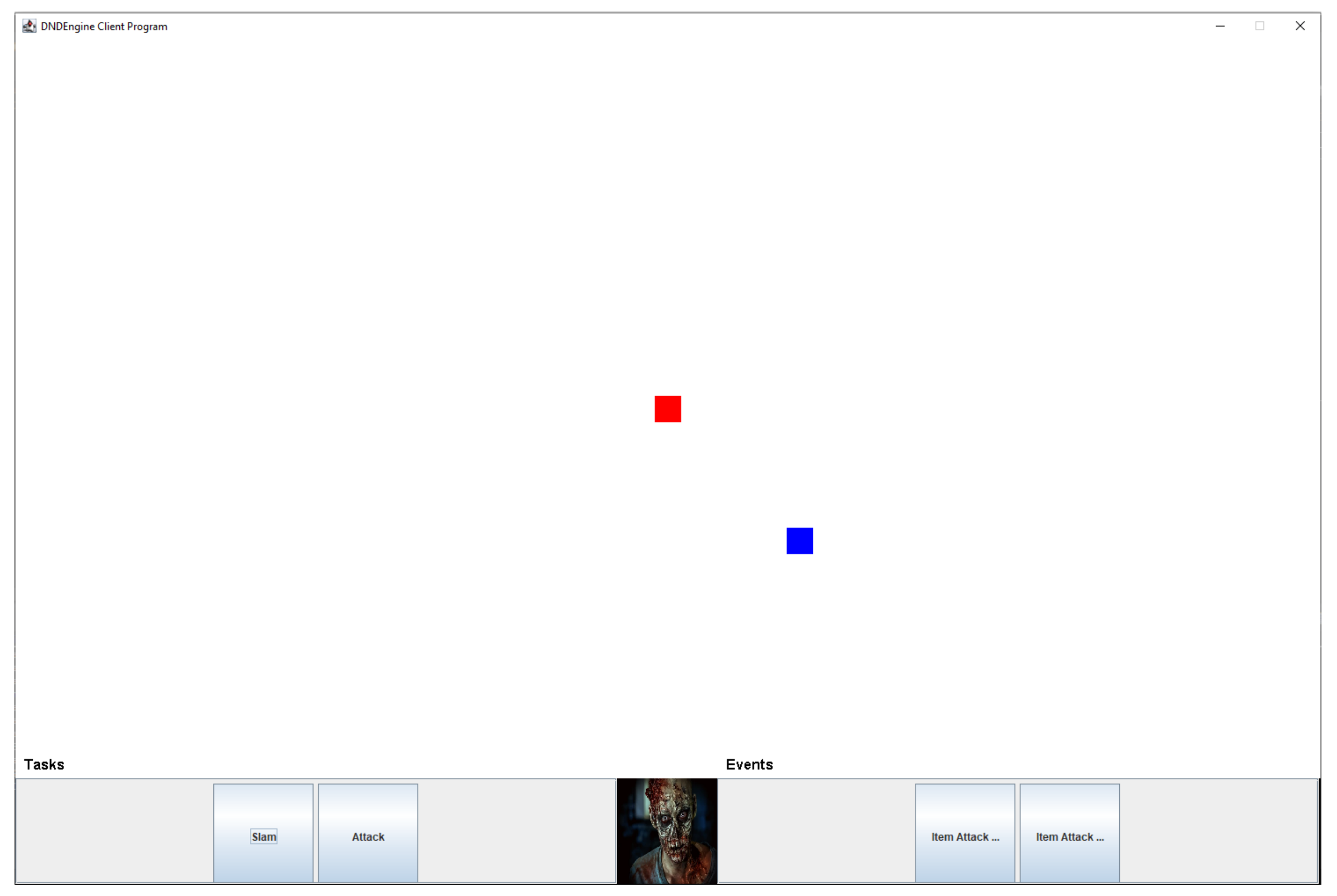


Figure 4: Client Graphical Interface
This client features two GameObjects on the game board (red selected, blue not selected).

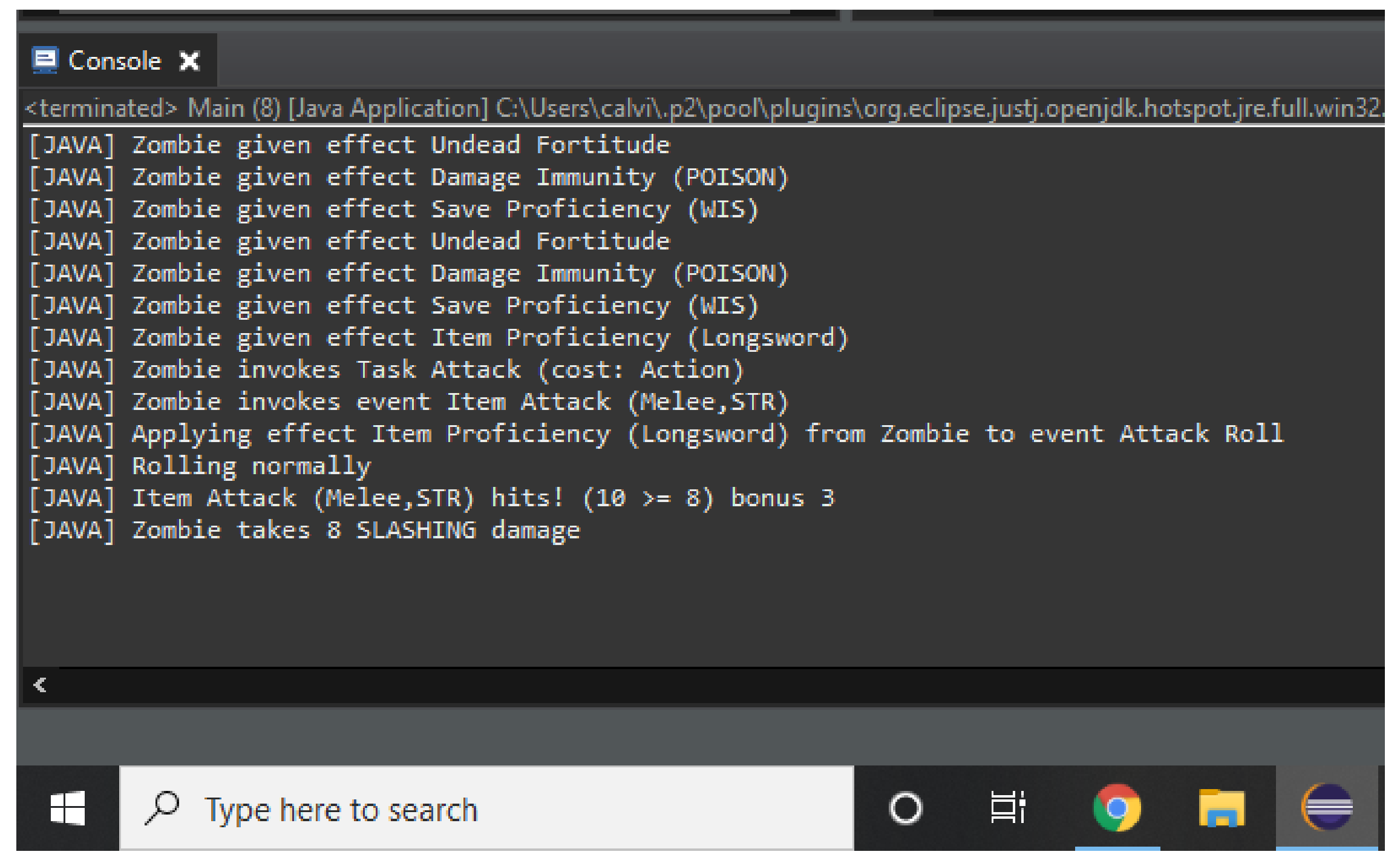
Interacting with Lua Scripts from Java

Lua is its own distinct programming language, and Java does not inherently know how to compile it. However, there is an open-source Java library called LuaJ which makes this interaction possible. By creating virtual machines which can process Lua scripts, LuaJ allows Java programs to function with logic and data defined by Lua scripts.

Each of the core data types are extensions of a Java class named Scriptable, which is dedicated to interacting with the LuaJ library. If a core data type class wishes to query the Lua script which it references, it must do so through the functions made available by Scriptable. In this way, if a different means of loading external logic is desired at some later time, it will involve minimal changes to the library architecture.

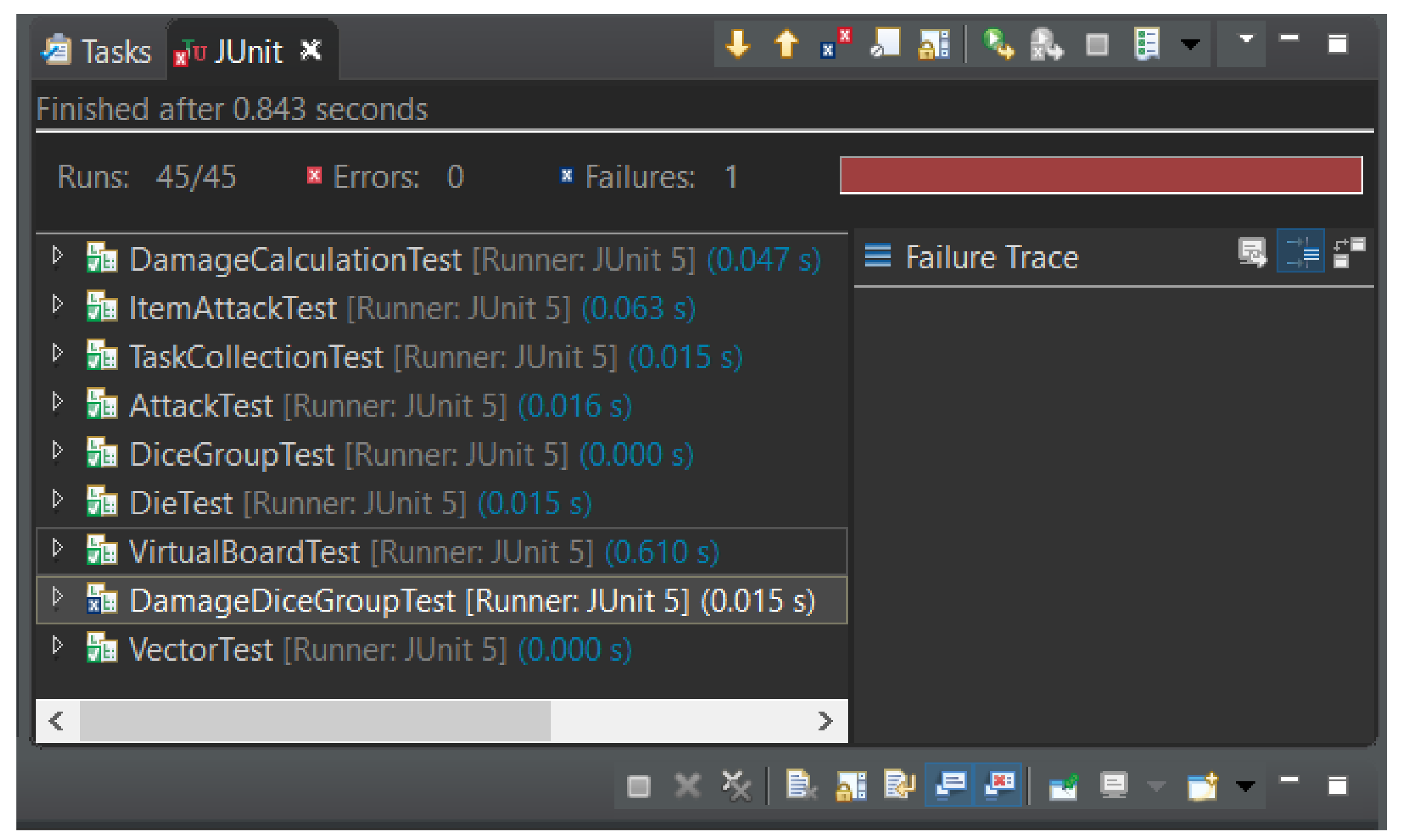
Library Architecture

The architecture of this library has evolved over the course of its development. Every architectural change has moved the project towards a more generalized, more abstracted collection of data types which give the client more power to define customized game content. The current library architecture can be seen in Figure 2 as an Entity Relationship (ER) diagram.



```
<terminated> Main (8) [Java Application] C:\Users\calvi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
[JAVA] Zombie given effect Undead Fortitude
[JAVA] Zombie given effect Damage Immunity (POISON)
[JAVA] Zombie given effect Save Proficiency (WIS)
[JAVA] Zombie given effect Undead Fortitude
[JAVA] Zombie given effect Damage Immunity (POISON)
[JAVA] Zombie given effect Save Proficiency (WIS)
[JAVA] Zombie given effect Item Proficiency (Longsword)
[JAVA] Zombie invokes Task Attack (cost: Action)
[JAVA] Zombie invokes event Item Attack (Melee,STR)
[JAVA] Applying effect Item Proficiency (Longsword) from Zombie to event Attack Roll
[JAVA] Rolling normally
[JAVA] Item Attack (Melee,STR) hits! (10 >= 8) bonus 3
[JAVA] Zombie takes 8 SLASHING damage
```

Figure 5: DNDEngine Console Output
Typical console output generated by the game library. This output is intended to eventually be recorded in log files for debugging purposes, but this feature has not been implemented yet. This output is not presented on the client software window as it is not intended to be directly viewed by the user.



```
Finished after 0.843 seconds

Runs:  45/45     Errors:  0     Failures:  1

DamageCalculationTest [Runner: JUnit 5] (0.047 s)
ItemAttackTest [Runner: JUnit 5] (0.063 s)
TaskCollectionTest [Runner: JUnit 5] (0.015 s)
AttackTest [Runner: JUnit 5] (0.016 s)
DiceGroupTest [Runner: JUnit 5] (0.000 s)
DieTest [Runner: JUnit 5] (0.015 s)
VirtualBoardTest [Runner: JUnit 5] (0.610 s)
DamageDiceGroupTest [Runner: JUnit 5] (0.015 s)
VectorTest [Runner: JUnit 5] (0.000 s)
```

Figure 6: Junit 5 Test Results
All test cases except for 1 pass. The one failure is due to a feature that is not yet implemented, but which is desired.

Developing an Example Client Interface

The current client interface renders a 2-dimensional view of the 3-dimensional game space, upon which each GameObject is drawn as a blue square. A GameObject may be selected by clicking on it. The selected GameObject is drawn as a red square so long as it is selected. Two horizontal scrollbars at the bottom of the screen display the Tasks available to the selected GameObject (left) and the Events queued by the selected GameObject (right). These scrollbars contain 1 button per Task or Event. When clicked, those buttons will invoke the appropriate Task or Event. However, the client currently has a bug which prevents the scrollbars from updating, so the GameObject must be deselected and then reselected to see any changes. The client interface does not implement most features made available by the most recent library code. This is due to a lack of experience developing graphical interfaces on the part of the developer, as well as the secondary nature of the client program in comparison to the library itself. A screenshot of the client can be seen in Figure 4, and Figure 5 shows example console output generated by the game library.

Testing the Library

As the client is insufficient for running manual tests, the library is tested by executing unit tests on its components with JUnit 5. Every feature of each data type can be tested in this way, as can the interactions between various different objects. The current suite of Junit tests does not cover the entire library, as it was implemented after the architecture of the project became sufficiently stable to support such tests. However, of the test cases which have been defined and which have been addressed by the library, every test passes without error. See Figure 6 for the results of the most up-to-date Junit tests.

RESULT & SUMMARY

To the extent that it has been developed and tested, the library functions correctly and without error. Although the client program does not utilize the complete suite of features made available by the latest library code, it demonstrates the functionality of the core behavior of the library. Further development of the client would make the full library available to the user. The library itself is currently incomplete, however. While it supports the core data types, certain phenomena which are iconic to Dungeons and Dragons such as casting spells are not yet supported. Further development is necessary before this project can boast that it automates all Dungeons and Dragons interactions.

The core data types identified by this project can model many games beyond just Dungeons and Dragons, such as Pokémon. The hard-coded derivations of the core data types intended for Dungeons and Dragons could just as easily be made to fit the theme of other popular games. If this project gains popularity, it might eventually branch out to provide similar automated interactions for other currently un-automated games. Visit https://www.youtube.com/watch?v=fWYXNlZpVIU for a short video presentation on this project.

References

[1] Gamma, Erich, et al. *Design Patterns Elements of Reusable Object Oriented Software*. 1st ed., Addison Wesley, 1998.

[2] Bechtold, Stefan, et al. *JUnit 5 User Guide*, 26 Oct. 2020, 22:06:33, junit.org/junit5/docs/current/user-guide/.

[3] Roseborough, James, et al. "Luaj/Luaj." *GitHub*, 1 Apr. 2020, github.com/luaj/luaj.