Asura Builder

Version 0.0.1

Authors

José Carlos Paiva *josepaiva94@gmail.com*
José Paulo Leal *zp@dcc.fc.up.pt*

Overview

Asura Builder is a Java framework for building Asura challenges (games), providing a game movie builder, a general game manager, several utilities to exchange complex state objects between the manager and the SAs, and general wrappers for players in many different programming languages. The framework is accompanied by a Command-Line Interface (CLI) tool (not yet finished) to easily generate Asura challenges and install specific features, such as support for a particular programming language, a default turn-based game manager, among others.

Getting Started

1. Install Java 8.
2. Install Maven. You can download it from here and follow installation instructions.
3. Use the Asura Game Maven Archetype to get a project for an Asura game.
4. Follow the instructions in the repository of the archetype to test, execute and package your game.
5. (just for JS) To execute Javascript, you need SpiderMonkey. After installed, update the pom.xml property `js.interpreter` to the executable of SpiderMonkey e.g.: /usr/bin/js52

You can also check this video to get started faster: https://youtu.be/licbuY_WK6M

Check this video to see how to preview the game: https://youtu.be/_OrJWcZxPFc

Structure of the Asura Builder Framework

```
java/pt/up/fc/dcc/asura/builder/
    base/
        exceptions/
```

```
            BuilderException          → General exceptions thrown by the Builder
            PlayerException           → Exceptions raised when a player is faulty
        messaging/
            Command                   → Commands sent from players to execute actions
            PlayerAction              → Wraps a command and a list of messages from a player to send to the Manager
            StateUpdate
        movie/                        → Contains the necessary utilities to build the game movie
            models/
                GameFrameItem
                GameMovie
                GameMovieFrame
                GameMovieHeader
                GamePlayerStatus
            GameMovieBuilder          →  Common interface for game movie builders
            GameMovieBuilderImpl      → Actual implementation of the game movie builder provided by Asura
        utils/
            CopyUtils                 → Copy/clone Java objects
            Json                      → JSON parser/writer to communicate with players and export the game movie
            JsonObject                → Object to be extended by other Java objects which can be written as JSON
        GameManager                   → Manages the game, given the players' processes.
        GameState                     → Interface that defines methods to manage the state of the game
    languages/                        → Contains language-specific classes to compile and execute players
        Language                      → Abstract class that defines the methods to compile and execute players
        JavaLanguage                  → Concrete implementation of a Language for Java
        JavascriptLanguage            → Concrete implementation of a Language for Javascript
        ...
    utils/                            → Utilities used in compilation and execution of programs
        FileUtils                     → Utilities for copying and locating files
    GameServer                        → Server that processes requests for visualizing the game
resources/
       wrappers/                      → Global wrappers that define methods to read and write JSON, log messages, ...
              <language>              → Folder for wrapper files of a language
```

Game Movie Builder

To facilitate the creation of graphical game-like feedback, Asura Builder introduces the concept of *game movie*, which consists of a set of frames, each of them containing a set of sprites together with information about their location and applied transformations, and metadata information, such as title, background, width, height, fps, and players (i.e., player names indexed by their ID). Furthermore, the concept of game movie is formally defined in a JSON schema. To allow the manager to easily build the JSON data, the game movie builder provides several methods, such as: setters for each metadata field, `addFrame()` which adds a frame to the movie, `addItem(sprite, x, y, rotate, scale)` which adds a sprite to the current frame in position `(x,y)` with the given rotation and scale, `addMessage(playerId, message)` which adds a message to the player, setters for observations and classification, and `saveFrame()/restoreFrame()` which allow to add a frame state to a stack and restore it later. **If a player breaks the rules of the game, `failedEvaluation(PlayerException e)` should be called before the game ends.**

The package `movie` contains all classes related to the game movie builder. Figure 1 presents the UML class diagram of this package. The authors' code uses only the methods defined in `GameMovieBuilder` to build the game movie, not needing to be aware of all the work behind.

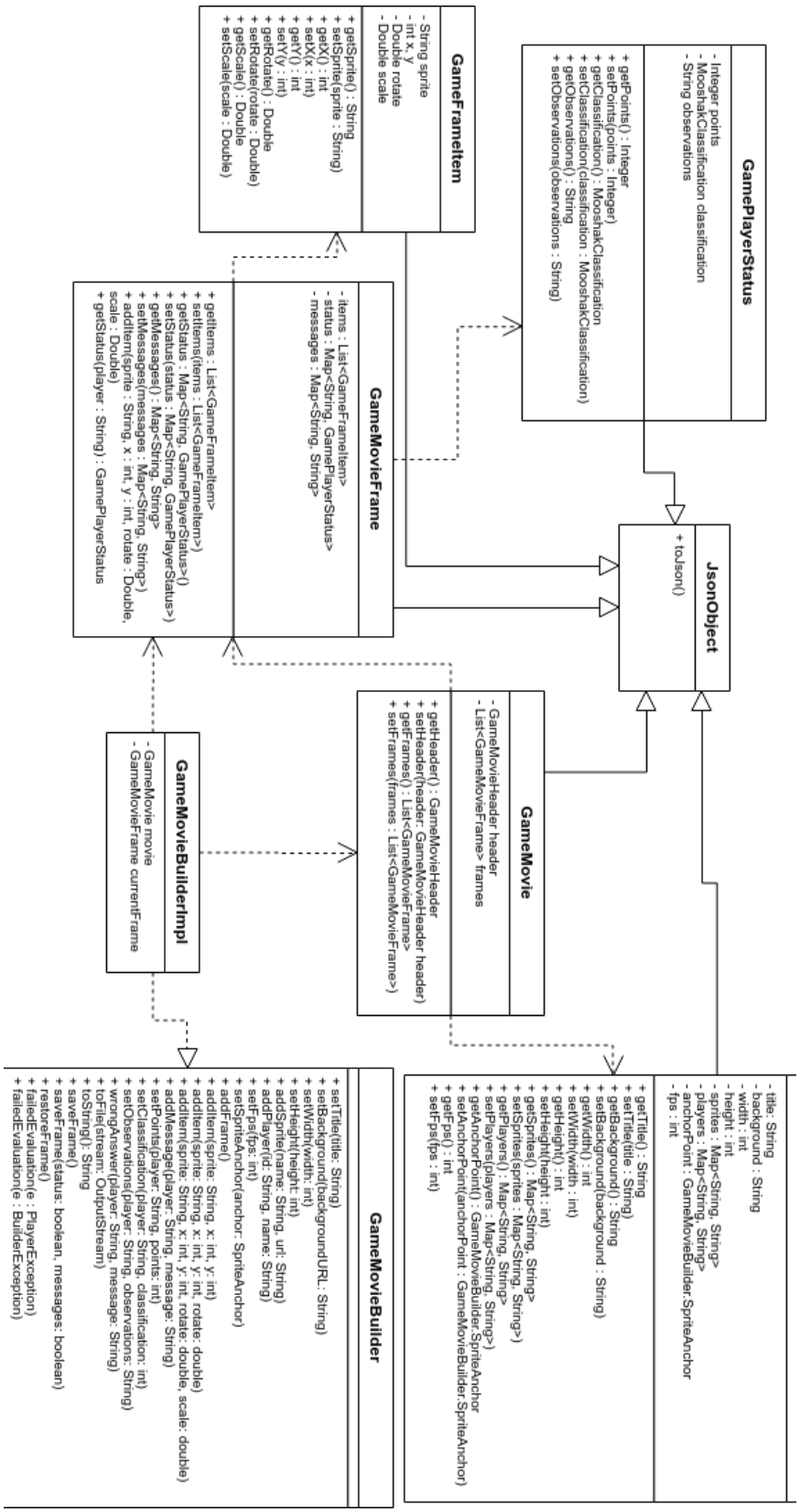

Figure 1: UML Class diagram of the movie package

Communication Manager-Players

The communication between the `GameManager` and the players is done through JSON or XML. The `GameManager` sends state updates to the players, updating them about changes to the state of the game. A state update contains a type, which identifies the object, and the object. The messages sent from the players to the `GameManager` contain the action (a command) that the player wants to execute as well as a list of messages for debugging purposes. Figure 2 presents the structure of the JSON objects that are exchanged during the game.

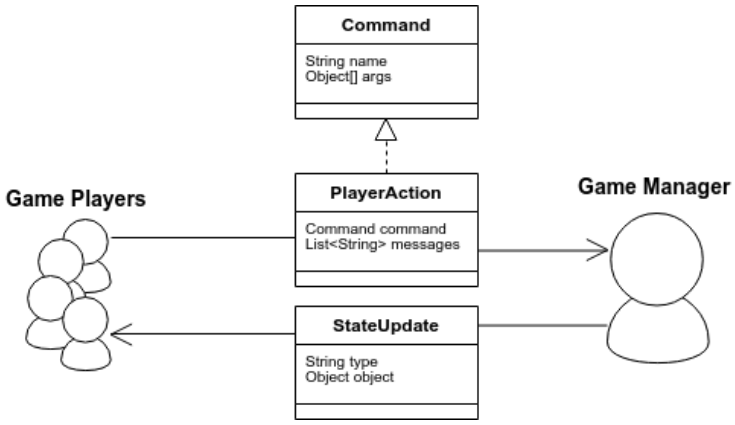

Figure 2: Diagram of communication between the manager and the player

Game Manager and Game State

The `GameManager` is the program that decides which player acts at each time. Besides that, it connects to the input and output stream of the players' processes to receive their actions and update them with changes on the game state. The state of the game is also managed by the `GameManager` based on the actions of the players. For that, state objects implement a common interface `GameState`, which allows `GameManager` to update it with the following methods:

> **`prepare(builder: GameMovieBuilder, players: Map<String, String>)`** which allows to initialize the state before the game starts.
>
> **`execute(builder: GameMovieBuilder, playerId: String, action: PlayerAction)`** that changes game state according to the action of a concrete player.
>
> **`getStateUpdateFor(player: String): StateUpdate`** which provides the object that needs to be sent to a certain player.

`endRound(builder: GameMovieBuilder)` which is called when all players' commands were executed in a round.

`isRunning()`: `boolean` which checks if the game is still running.

`finalize(builder: GameMovieBuilder)` that runs any necessary tasks when the game has ended.

Most of these methods receive a game movie builder as parameter, allowing the state to manage the movie, reflecting any changes made to it.

The `GameManager` and `GameState` are specific to each game. Therefore, authors must extend the `GameManager` with the specific rules of their games and implement `GameState` to update the state and movie according to the game.

Wrappers

Wrappers are sets of functions that aim to provide players with an higher level of abstraction, so that they can focus on solving the real challenge instead of processing I/O. There are two types of wrappers: global and game-specific wrappers.

Global wrappers are defined by the Asura Builder framework. They implement functionality that is common to players of any game in that language. This includes methods to read and write JSON, log messages, and call the functions on the abstract and concrete players that implement the game/player-specific functionality.

Game-specific wrappers “extend” global wrappers implementing functions that allow to process state updates, get and set values of the state directly, and send actions. Besides that, they implement the player lifecycle, reading and writing sync.

Wrappers are language-dependent. Currently, there are two languages with global wrappers: Java and Javascript (ES5 and ES6). They can be found here.

Generated Project for Asura Games

After running the generate from archetype command, a project completely structured for the development of Asura games is created. Instructions on how to run, test, and package are present here.

Structure

```
java/<PackageName>/
        <GameName>Manager              → Concrete manager of the game
        <GameName>State                → Implementation of the game state
resources/
        <ArtifactId>/
```

```
            skeletons/          → Folder to store skeletons for software agents developed by the users
            solutions/          → Folder to store software agents developed by the author
            wrappers/           → Folder to store game-specific wrappers organized by language
                    <language>
      images/                   → Folder to store images of the game
test/java/<PackageName>/        → Tests for checking game functionality
      <GameName>ManagerTest
      <GameName>StateTest
```