Animal Chess / Djungle

Maximilian Münzer, Phuoc Thang Le, and Anna-Maria Dickmann

University Hamburg Informatik, Hamburg, Germany

Abstract. This is a developer handbook which describes the key functions of the Djungle game based on Racket's 2hdtp/universe library.

Keywords: Racket · Djungle · Board game

1 Overview

Welcome to the developer's handbook for the game Animal Chess, also known as Djungle. This documentation's purpose is to increase understanding of the source code and outline further potentials. The rules of the game and game launch requirements can be found in the additional player's handbook. To test the server and simulate a multiplayer game its necessary to type in the ip "DEBUG" in the starting window.

2 Architecture

The Djungle game consists of three key modules. The client, network and server module.

- The client module contains the game logic and the graphical functions for rendering purposes.
- The network module contains the functions to encode the worldstate of the game in a transferable message for the server and decodes a receiving message in a worldstate.
- The server module contains the functions to handle the connection between clients. The server itself contains no game logic except the checks for a game ending condition.

The modules depend on the additional files helper-function.rkt, Structs.rkt and Constants.rkt. For clarity and further maintenance/development these are individual files. The diagram 2 provides an overview of the game. The key modules are highlighted and modules are described by their functionality.

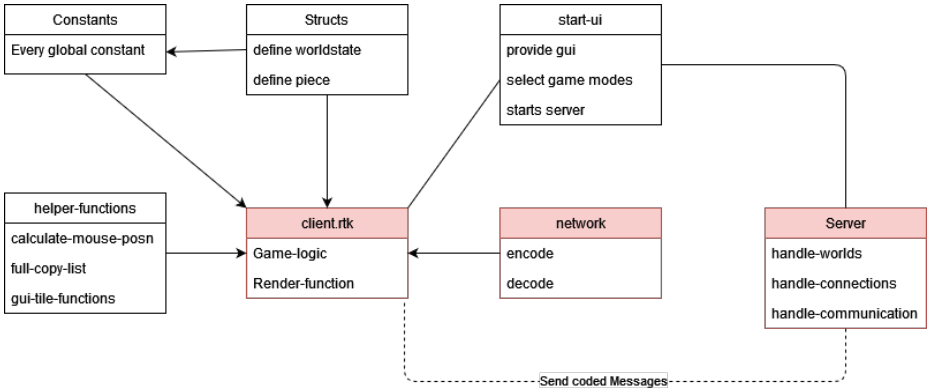

Fig. 1. Simplified module structure with described functionality. An arrow means this module provides functions for the module

The program uses the big-bang function from the 2htdp/universe library. Big-bang offers different handlers to control the game, including a event handler for mouse events. The game is based on mouse events. The following diagram 2 outlines the flow of the clients architecture.

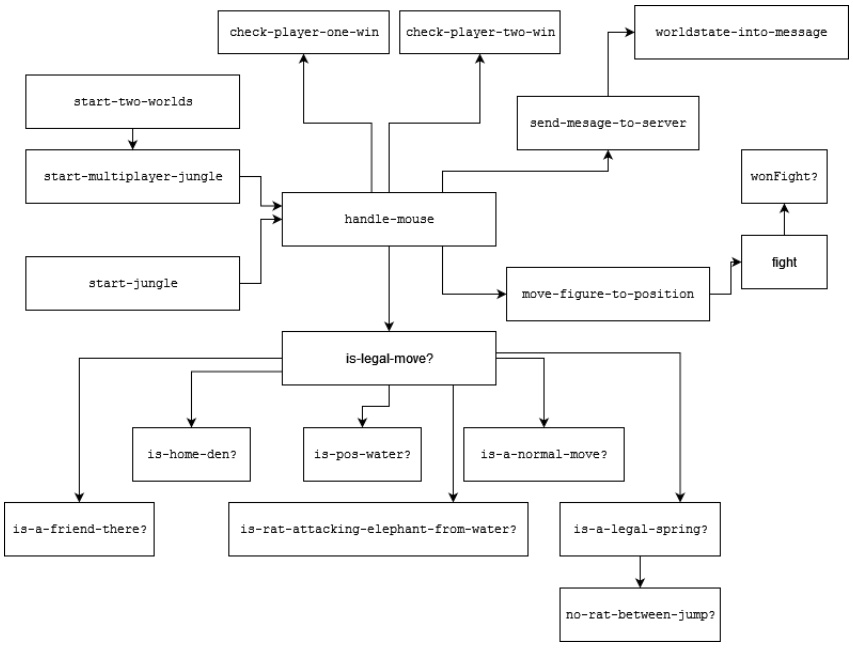

Fig. 2. Simplified client structure with important functions

3 Toolkit Racketlang / Used Libraries

The entire animal chess game was built with the Racket Lang ecosystem version 8.2, available under https://download.racket-lang.org. It uses the core package lang racket, the additional packs lang/posn, racket/include, racket/gui/base and the additional teachpacks 2htdp/universe and 2htdp/image.

4 Application

The application is split into two directories. The root directory contains all source-code to run the application and required images are stored in /Images. The program's main functions are defined in the client.rkt, message.rkt and server.rkt file.

4.1 Global settings and variables

The game's general definitions and settings such as board and tile-size, image and player color assignments are assigned in the Constants.rkt file. If a procedure requires to be made available globally it should be defined here for easier maintenance. An exception is the scale-factor for tiles with images which needs to be declared in respective files to avoid circle definitions. All global procedures are exported via all-defined-out.

4.2 Game board

The Animal Chess game board has 7 x 9 tiles as defined in Constants. Each tile is assigned a position coordinate on the board via a call to the procedure make-posn. To use make-posn the lang/posn package is required. Each position represents a range of respective x and y coordinates that describe a square on the game board. Their exact size is determined via the constant TILE-SIZE. The helper-functions.rkt provides a method to calculate real mouse positions from pixel based coordinates to posn based coordinates. Every position, tile or piece is represented by a posn-equivalent between (0,0) and (7,8).

4.3 Structs

In Racket structs are user defined structure data types. A call to a structure procedure contains a value of each previously with struct defined field. Animal Chess has two structures defined in structs.rkt.
The struct *piece* is responsible for all figures on the board. It contains the parameters name color position rank image and selected-image which are used to store the following information:

- name: a string to identify the piece, corresponding to the respective animal name

- color: passing designated player's color via PLAYER1 or PLAYER2 constant
- position: call to the make-posn procedure from lang/posn to assign game token's initial position on the board grid
- rank: each token is assigned a unique number between 1 and 8 according to the game rules found in the player's handbook. These number define his combat rank.
- image: a procedure with an image as defined in Constants.rkt to visualize the respective animal is passed
- selected-image: a procedure which assigns the image of a currently active token is used

The second struct *worldstate* is our game's worldstate. The three parameters pieces, selected, turncolor can be assigned and represent the following information:

- pieces: list of current tokens on the board to determine the currently alive ones.
- selected: actively selected piece
- turncolor: color of player whose turn is next

4.4 Server communication

Animal chess offers two different play modes: local and network. The *server.rkt* file contains all information necessary for distributed network play.
Information between parties is transmitted via a list representing the game board. It is not possible to directly message a struct to a server. Therefore it is necessary to implement a encoding/decoding function to convert the worldstate into a transferable message. The function worldstate-into-message from the network.rtk file encodes the worldstate into a list representation of strings. For example the following code represents an excerpt of the initial worldstate:

```
(list "red" "rat" "red" 0 2 "cat" "red" 5 1 "dog" "red"
1 1 "wolf" "red" 4 2 ...)
```

The list starts with a color corresponding to the next player's turn and is followed by each piece's state encoded as a four tuple of its name, color and its two posn coordinates. Furthermore the complete message to the server contains another string to represent a event, i.e. whether the game is won or its your turn. The currently used strings are: "wait", "turn", "won" and "lost". A complete message between the server and the client contains an "event string" and a board representation encoded in a list of strings.
The procedure empty_board initializes the worldstate used by universe. The function handle_message handles the updating of local clients between the servers. To prevent message corruption it checks for every received message message if it's a list and divisible by four. Each state is immutable and thus every time the board requires an update a full new copy is created and transmitted.

4.5 Handling worldstate

The game's worldstate is handled via the 2htdp/universe package in particular the big-bang function.
Important annotation the complete world state is a list of a string and the struct worldstate. The first string determines if the game is started locally or if its your turn/win/loose in a multiplayer game. It represents the event message from the server.
In local mode, initially, the first instance of world state is created as (list "local" (world state alive null PLAYER1)). The render function will draw accordingly to the state by calling the function tiles+traps+figures+highlight.
With the information provided by the list "alive", the list of alive pieces will be drawn on board with their given location. The highlight function will receive the argument "selected" and highlight accordingly all the possible move on the map.
Whenever an event happens in the game e.g a new piece is selected or removed from the list "alive", a new world state is created and passed to render function in universe. Hence, the board will be completely redrawn.
In multiple-player mode, a world state instance will be translated in form of a string for the communication within the network. When received, the message is decoded and realized locally as a world state.

4.6 Implementation of game rules

Game rules are implemented in Client.rkt in the section *legal move rules*. The procedure *is-legal-move?* calls the other rule procedures to check whether certain conditions according to the game rules are satisfied. Depending on whether they are true a respective Boolean value is returned.

5 Future Prospects

Currently the game is fully playable, however, there are some possible future features that can enhance the game play experience. For example a feature that players can adjust the board size via a GUI dialog. The already implemented scaling factor allows players to adjust tile sizes via source code. As for further graphics the win and lose screens can be adjusted for distributed play, currently they are only properly available in local mode.
Another great improvement can be done on the server side. At the current state of development it is not possible to reconnect to a game session or even directly choose an opponent. A new feature can be a responsive message for the connection process, as currently the user does not get any feedback if a connection is pending or failed.