System Design Document for the Challenge Accepted project, (RAD)

- *A digital version of 'Upp till Bevis'.*

Contents

Version: 2.0
Date: 2013-05-27

Authors:

Isabelle Frölich	900831-2846
Madeleine Appert	891110-4845
Johan Gustavsson	871024-7134

This version overrides all previous versions.

1. Introduction

1.1. Design Goals

The bond between our GUI and our domain should be as loose as possible, to facilitate the possibility to exchange the GUI for another. It should also be easy to isolate th GUI from the domain in the case of wanting to test the domain solely.
For usability see RAD.

1.2. Definitions, acronyms and abbreviations

- GUI, graphical user interface , the visual part of the application.
- Java, platform independent programming language.
- JRE, the Java Run time Environment. Additional software needed to run a Java application.
- Host, the computer that the game will run on.
- Team, a team consists of at least two people competing together, against other teams.
- Mission, tasks every team has to manage in order to move forward.
- Turn, the turn for each team. A team can only make active decisions during their turn, but can be affected by the turns of other teams.
- Normal Tile, not a challenge tile.
- Normal Turn, the type of turn initiated when a team stands on a normal tile.
- Challenge, the type of turn when a team stands on a white tile and will compete with another team.
- Cha, an abbreviation for Challenge Accepted.

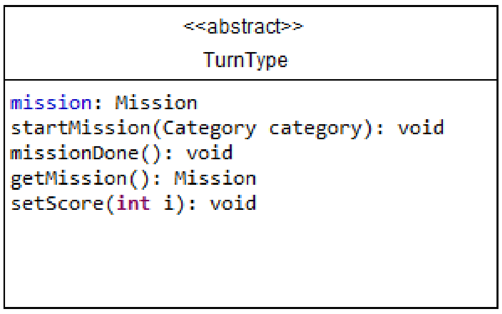
2. System Design

2.1. Overview

The application is using a so called 'event bus'. See point 2.1.4 (Event Handling) for more information.

2.1.1. Model functionality

The functionality is not greatly partitioned into several interfaces etc. There is an abstract class which hold the common functionality of different types of Turn. Please see figure 1.



```
mission: Mission
startMission(Category category): void
missionDone(): void
getMission(): Mission
setScore(int i): void
```

Figure 1.

2.1.2. Rules

The rules are fixed and listed in the application for the teams to see during any part of the game.

2.1.3. Unique Identifiers

We will not use any globally unique identifiers for any entity.

2.1.4. Event handling

In order to keep our domain and our GUI flexibly connected, and to be able to isolate and restrict the scope affected by an event taking place, we have created an 'event-bus'. The event-bus completely takes care of receiving, transmitting and sending any event which requires action.

2.1.5. Internal Representation of text

All text in the application can be found within the code, and the classes which need access to the text.

2.2. Software decomposition

2.2.1. General

The application is decomposed to the following modules (see Figure 2):

- cha, is our main package.
- cha.gui, main GUI for application.
- cha.domain , is the core of the application.
- cha.event , classes for the event bus.
- Main , is the class holding the main-method, the entry point of the application.

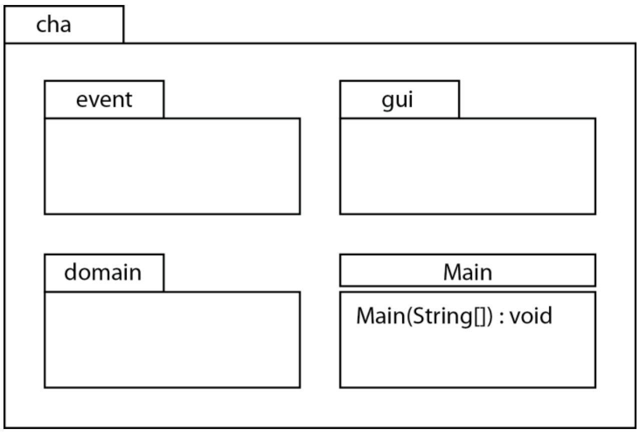


Figure 2. High Level Design.

2.2.2. Decomposition into subsystems

There are no subsystems in our packages.

2.2.3. Layering

The layering is as indicated in Figure 3 below.

2.2.4. Dependency Analysis

The dependencies are as indicated in Figure 3 below. There are no circular dependencies .

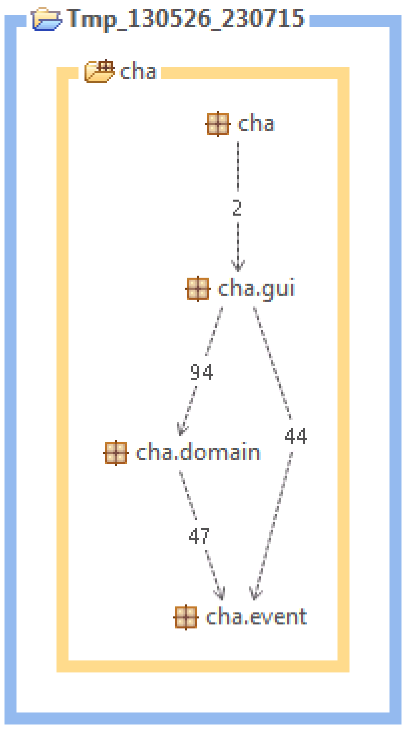


Figure 3. Layering and dependency analysis.

2.3. Concurrency Issues

This is a single threaded application. Everything will be handled by the Swing event thread.

2.4. Persistent data management

N/A Because we do not save anything from our game.

2.5. Access control and security

N/A.

2.6. Boundary conditions

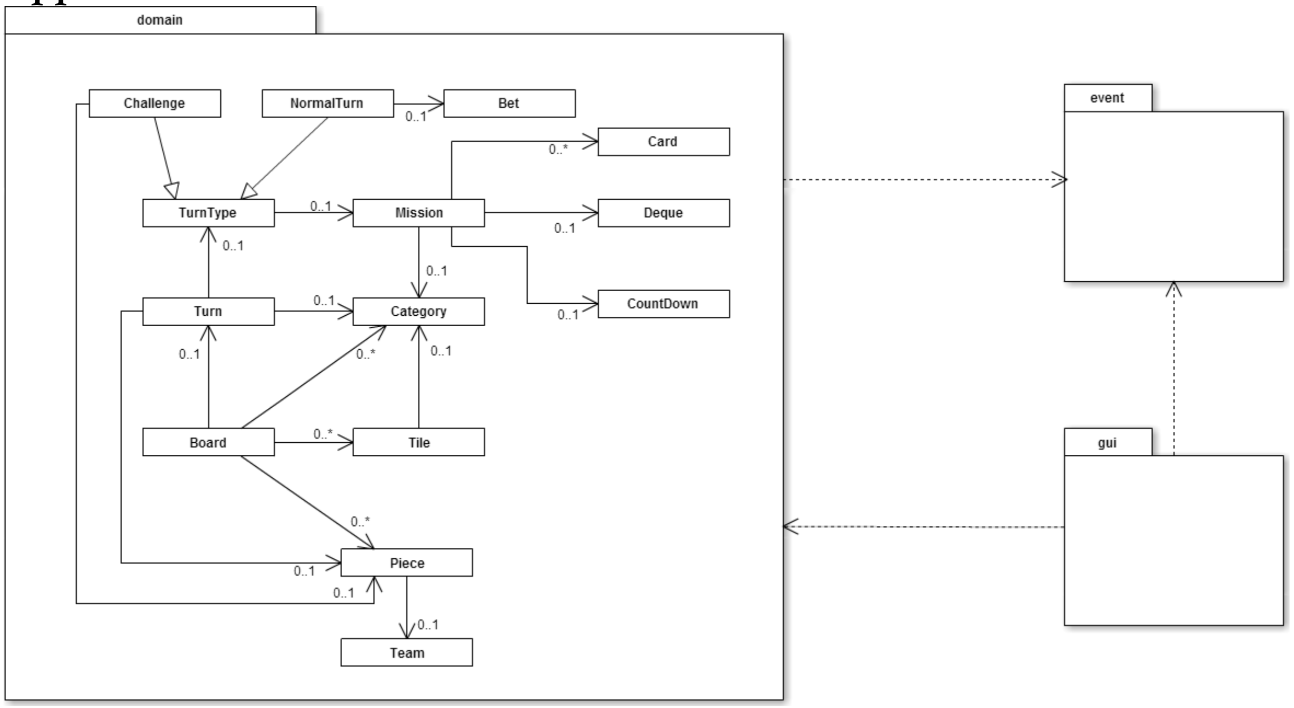
Application launched and exited as normal application.

3. References

General explanation of the game (in Swedish).
http://www.braspel.com/?id=317

Thorough explanation of game rules (in Swedish).
http://www.braspel.com/filearchive/1/1917/rules_UTB%20new.pdf

4. Appendix



- Board contains all of the tiles with Categories.
- Bet keeps track of how many missions the active team is betting to manage.
- Card contains the text with the mission or answers explained, for the current mission.
- Category is an enum class that keeps track of which categories are available.
- Challenge is the child class of TurnType that controls when two teams shall compete against each other.
- CountDown keeps track of the time limit of 30 seconds which is the maximum amount of time for a mission.
- Deque contains all of the different possible cards that can be chosen.
- Mission controls the current mission and which cards are shown.
- NormalTurn is also a child of TurnType and controls all normal turns.
- Piece is every teams representation on the board that keeps track of the teams positions.
- Team is the different teams existing and their names.
- Tile, the tiles on the board with different categories or challenges.
- Turn, controls whose turn it is, what kind of TurnType it is, the teams bet and the missions.
- TurnType is the parent class to Challenge and NormalTurn.