System design document för spelet Chans

Måns Josefsson, Felix Holmesten, William Andersson, Vilhelm Hedquist och Oscar Johansson

02/10/2020 - Version 1

1 Introduction

The project consists of a game that can be compared to the boardgame Risk. The game does have a Chalmers theme with a map with different places on campus that the players can take over.

1.1 Definitions, acronyms, and abbreviations

Risk:
A classic board game, which revolves around conquering the world and battle against opponents for each other's territories using dice. (For more information: https://en.wikipedia.org/wiki/Risk_(game))

Chalmers Chance:
The name of the game.

Student division:
A student division in the context of the game is a local organization of Chalmers student union, where each of the 16 student divisions are connected to a unique study programme.

2 System architecture

The normal flow of operation for the application is as follows: The first scene in the GUI is the start menu. There the user can choose to either start the game or quit the game, the usual choice that the user takes is to start the game. The next scene is a scene where the users set up the game, the users choose how many players that's gonna play the game and then moves on to click the button "Start Game". The next scene is the map and all the spaces. The game then goes on to the three phases that a player can do in one round. The first is "Deploy" where the user deploys units. The second one is "Attack" where the user attacks another space. The third one is "Movement" where the player gets to move units. That is one round for one player and then the turn goes over to the next player in line. The game is won when one player owns all spaces.

Except for the normal flow there are also some more scenes in the game. There is a pause menu where the players can quit the game which terminates the application, start a new game or quit to the start menu.

There are three important packages in the project that controls the application. Model, view and controller. Model holds the game loop and all the data in the project. View is currently only one class called MapView. MapView is creating the different scenes, colors, texts and everything GUI related. The controller is the biggest package, it controls when the different scenes are supposed to change and what's supposed to happen when clicking the different buttons but mostly, the controller controls the view.

3 System design

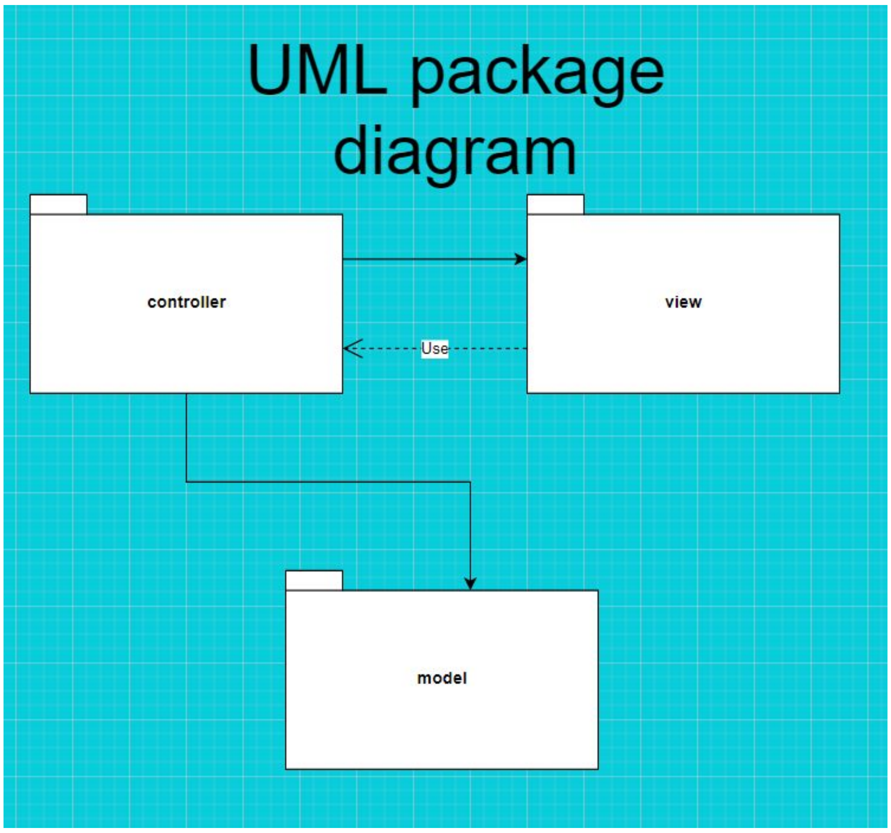

Figure 1: UML package diagram from the application

As mentioned in the previous paragraph the project is using an MVC structure. However because most of the changes are happening when the users click on the buttons (and spaces) it was decided to change it up a little. As of right now the controller depends on the view and the model, but the controller is also the one to initiate change in the view. The view has a dependency to the controller as well. The most important thing to note is that the model doesn't have any dependencies to the other packages. This means the model can be used as a stand alone package, with other controllers and views. The controller is the biggest package in the project and holds all the classes that make use of button actions and GUI as the game is played.

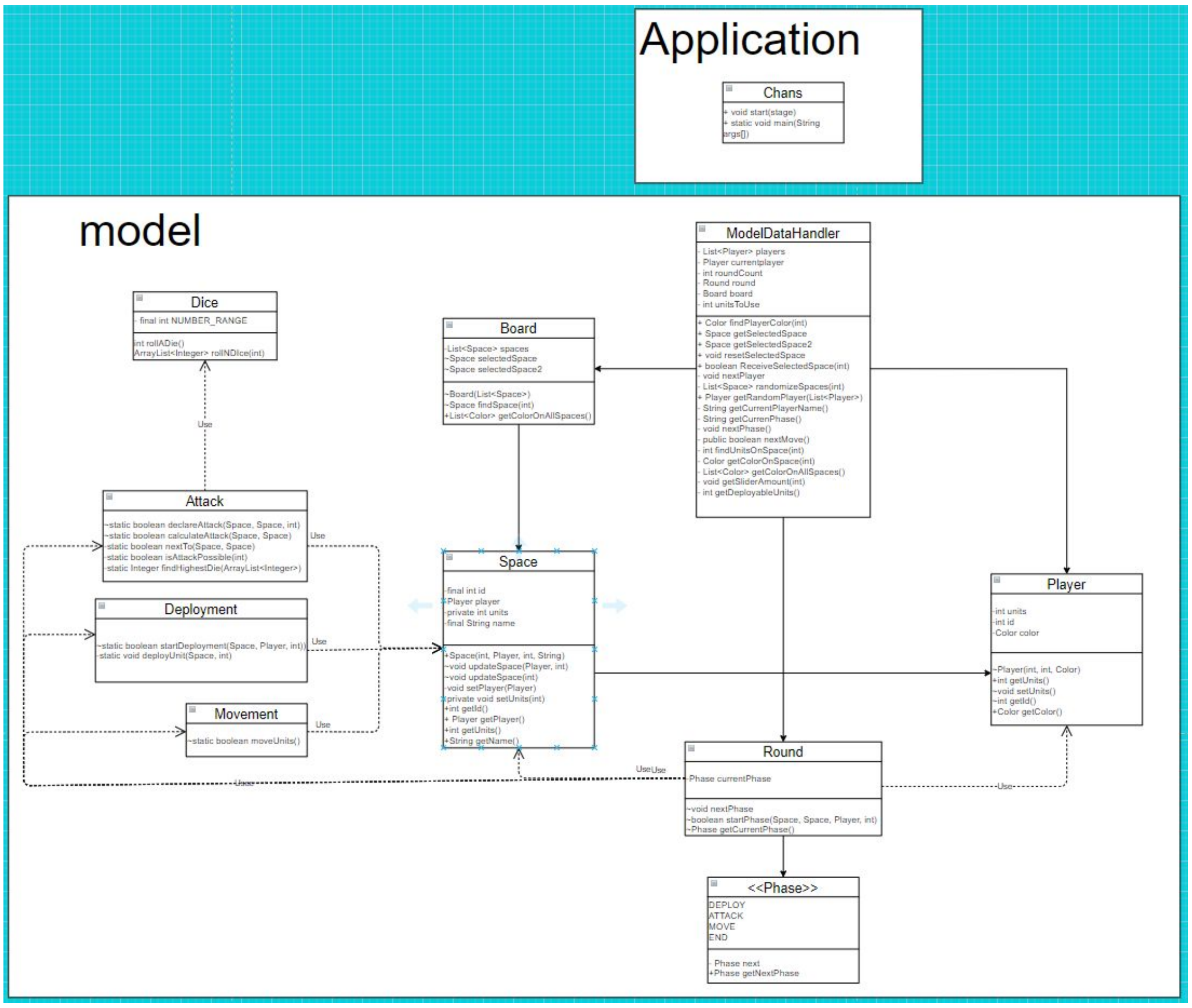
Application
Chans
+ void start(stage)
+ static void main(String args[])
model
Dice
- final int NUMBER_RANGE
int rollADie()
ArrayList<Integer> rollNDice(int)
Use
Attack
~static boolean declareAttack(Space, Space, int)
~static boolean calculateAttack(Space, Space)
-static boolean nextTo(Space, Space)
-static boolean isAttackPossible(int)
-static Integer findHighestDie(ArrayList<Integer>)
Use
Deployment
~static boolean startDeployment(Space, Player, int)
-static void deployUnit(Space, int)
Use
Movement
~static boolean moveUnits()
Use
Use
Board
-List<Space> spaces
~Space selectedSpace
~Space selectedSpace2
~Board(List<Space>)
~Space findSpace(int)
+List<Color> getColorOnAllSpaces()
Space
-final int id
-Player player
-private int units
-final String name
+Space(int, Player, int, String)
~void updateSpace(Player, int)
~void updateSpace(int)
-void setPlayer(Player)
-private void setUnits(int)
+int getId()
+ Player getPlayer()
+int getUnits()
+String getName()
ModelDataHandler
- List<Player> players
- Player currentplayer
- int roundCount
- Round round
- Board board
- int unitsToUse
+ Color findPlayerColor(int)
+ Space getSelectedSpace
+ Space getSelectedSpace2
+ void resetSelectedSpace
+ boolean ReceiveSelectedSpace(int)
- void nextPlayer
- List<Space> randomizeSpaces(int)
+ Player getRandomPlayer(List<Player>)
- String getCurrentPlayerName()
- String getCurrenPhase()
- void nextPhase()
- public boolean nextMove()
- int findUnitsOnSpace(int)
- Color getColorOnSpace(int)
- List<Color> getColorOnAllSpaces()
- void getSliderAmount(int)
- int getDeployableUnits()
Player
-int units
-int id
-Color color
~Player(int, int, Color)
+int getUnits()
~void setUnits()
~int getId()
+Color getColor()
UseUse
Round
-Phase currentPhase
~void nextPhase
~boolean startPhase(Space, Space, Player, int)
~Phase getCurrentPhase()
Use
<<Phase>>
DEPLOY
ATTACK
MOVE
END
- Phase next
+Phase getNextPhase

Figure 2: Dependencies in the model package.

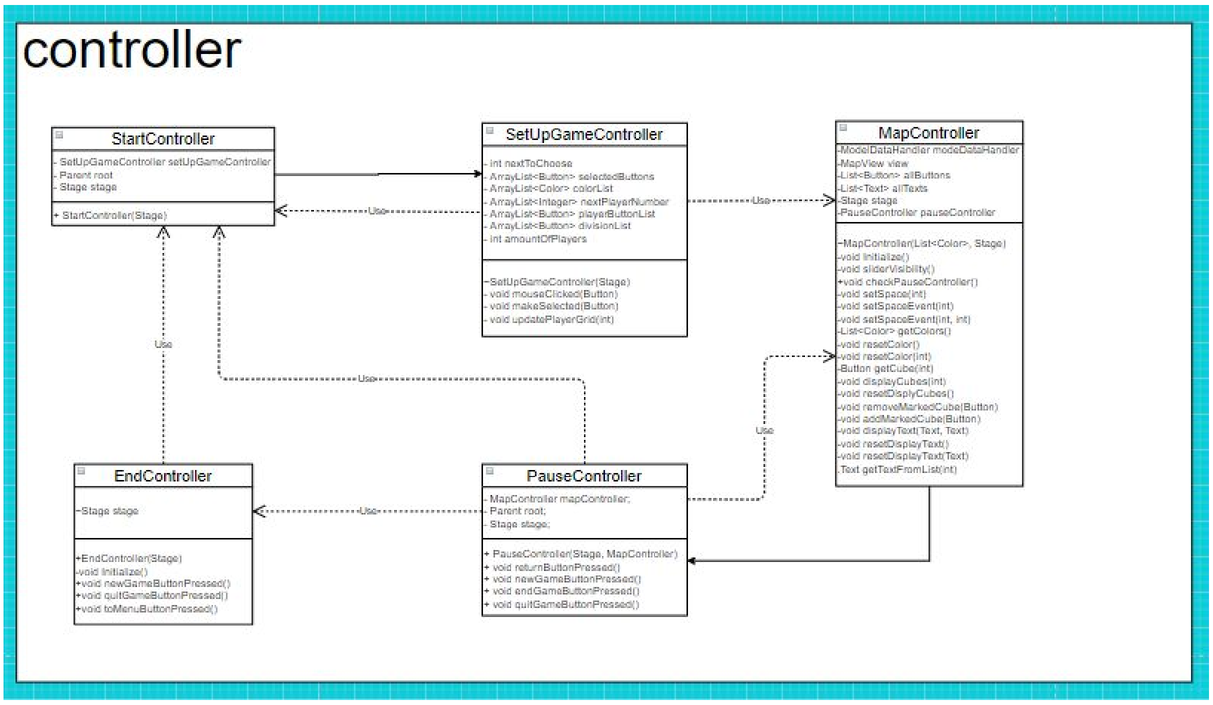
controller
StartController
- SetUpGameController setUpGameController
- Parent root
- Stage stage
+ StartController(Stage)
Use
SetUpGameController
- int nextToChoose
- ArrayList<Button> selectedButtons
- ArrayList<Color> colorList
- ArrayList<Integer> nextPlayerNumber
- ArrayList<Button> playerButtonList
- ArrayList<Button> divisionList
- int amountOfPlayers
~SetUpGameController(Stage)
- void mouseClicked(Button)
- void makeSelected(Button)
- void updatePlayerGrid(int)
Use
MapController
-ModelDataHandler modeDataHandler
-MapView view
-List<Button> allButtons
-List<Text> allTexts
-Stage stage
-PauseController pauseController
~MapController(List<Color>, Stage)
-void initialize()
-void sliderVisibility()
+void checkPauseController()
-void setSpace(int)
-void setSpaceEvent(int)
-void setSpaceEvent(int, int)
-List<Color> getColors()
-void resetColor()
-void resetColor(int)
-Button getCube(int)
-void displayCubes(int)
-void resetDisplyCubes()
-void removeMarkedCube(Button)
-void addMarkedCube(Button)
-void displayText(Text, Text)
-void resetDisplayText()
-void resetDisplayText(Text)
.Text getTextFromList(int)
Use
Use
Use
EndController
~Stage stage
+EndController(Stage)
-void initialize()
+void newGameButtonPressed()
+void quitGameButtonPressed()
+void toMenuButtonPressed()
Use
PauseController
- MapController mapController;
- Parent root;
- Stage stage;
+ PauseController(Stage, MapController)
+ void returnButtonPressed()
+ void newGameButtonPressed()
+ void endGameButtonPressed()
+ void quitGameButtonPressed()

Figure 3: Dependencies in the controller package.

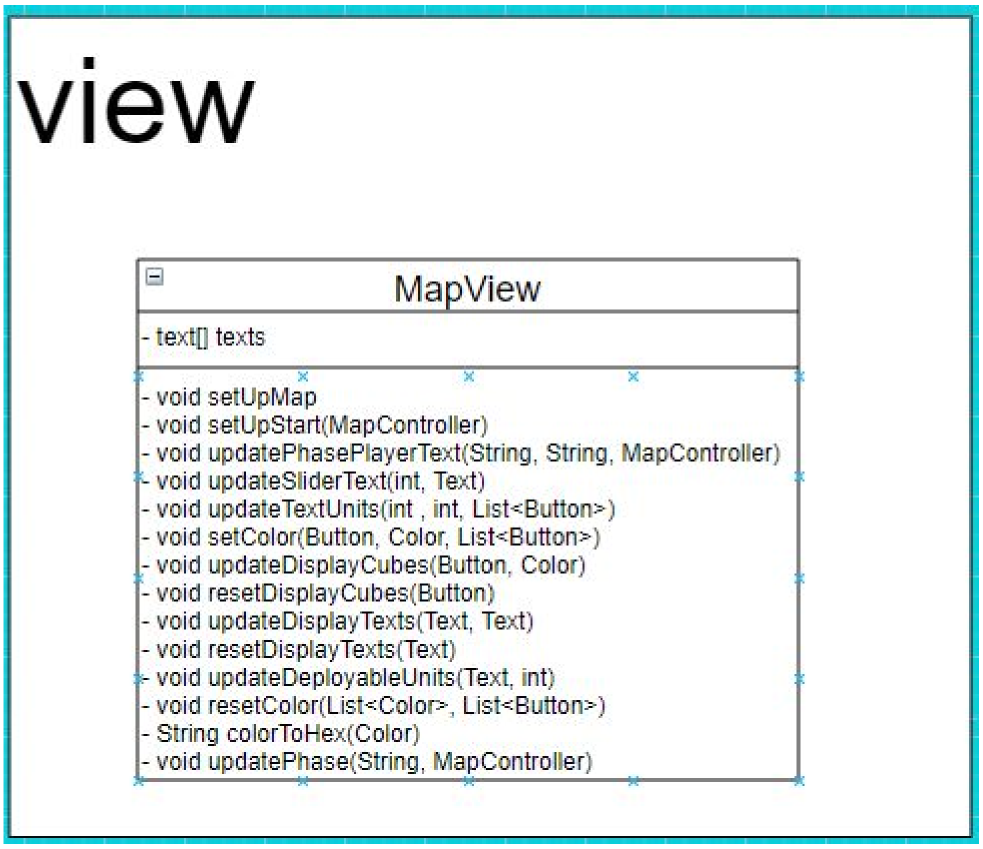
view
MapView
- text[] texts
- void setUpMap
- void setUpStart(MapController)
- void updatePhasePlayerText(String, String, MapController)
- void updateSliderText(int, Text)
- void updateTextUnits(int , int, List<Button>)
- void setColor(Button, Color, List<Button>)
- void updateDisplayCubes(Button, Color)
- void resetDisplayCubes(Button)
- void updateDisplayTexts(Text, Text)
- void resetDisplayTexts(Text)
- void updateDeployableUnits(Text, int)
- void resetColor(List<Color>, List<Button>)
- String colorToHex(Color)
- void updatePhase(String, MapController)

Figure 4: The current extent of the view-package.

ModelDataHandler is the central point to access the model package and Chans initiates the aplication by calling StartController.

A clear similarity between our domain model and design model is that they contain the same classes and the dependencies between the classes stay consistent.

Design Patterns and UML sequence diagram to be added in a later stage.

4 Persistent data management

The only persistent data that is used in the application is images, which are stored in a package within the source code.

5 Quality

The application is tested with JUnit-tests. All public or package private methods should be tested with at least one test. The tests can be found in the test-package.

List of all known issues:

- All Java and XML files currently do not have what is stated below.
  NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.
- Analytical tool to show dependencies (eg. STAN or similar) hasn't been used yet.
- Analytical tool to show quality (eg. PMD) hasn't been used yet.
- All public and package private methods haven't been tested.
- Some background color/text color combinations are currently difficult to read.

6 References

JavaFX - Library used to create the GUI of the application. (See: https://openjfx.io/)

Scene Builder - Integrated with JavaFx and used to design the GUI with simple Drag and Drop features. (See: https://gluonhq.com/products/scene-builder/)

IntelliJ IDEA - IDE used for programming. (See: https://gluonhq.com/products/scene-builder/)