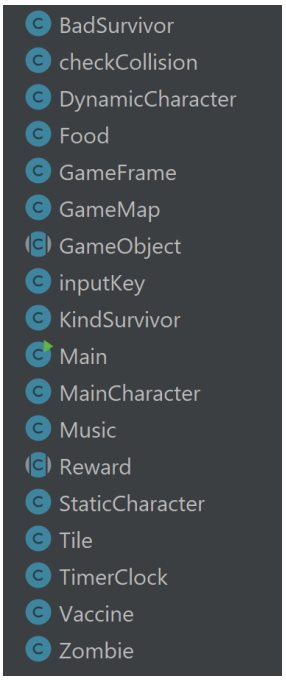
CMPT276 D100 - Assignment 3 (April 11, 2022)
Risa Kawagoe & Anika Sheikh (Group16)

Report (max 2 pages)

The classes we implemented including partial implementation



Risa Kawagoe:

- Music (all)
- inputKey (partially: title screen, tutorial screen, narration screen, end screen)
- GameFrame (partially: run method, paint component for tutorial state, narration state, and title state, change level state)
- GameMap (partially: map level selection depending on input)

Anika Sheikh:

- GameMap (constructor, getOriginMap, getStartPoints)
- checkCollision (all)
- GameFrame (partially: run mehtod, timer)
- TimerClock (all)
- MainCharacter(constructor, SetDefaultValue, getMCImages)
- KindSurvivor(partially speak(), drawKindCharcMsgBox)

Smells of anti-patterns

1. lack of documentation
    a. inputKey - Method keyPressed() is a large method with little documentation so it gets a bit confusing -> solution: break down the method using helper functions within the class
2. code duplication

a.

```
                    gf.commandNum = gf.numOfCommands - 1;
                }
            }else if(key == KeyEvent.VK_ENTER) {
                gf.playSoundEffect(0);

                System.out.println("enter pressed in change level screen");
                if(gf.commandNum == gf.levelEasy) {
                    System.out.println("change to easy");
                    gf.numOfVaccines = 5;
                    gf.gameLevel = gf.levelEasy;
                    gf.mc.setDefaultValue(gf.tileFrame.getStartPoints(gf.levelEasy));
                    gf.gameState = gf.titleState;
                }else if(gf.commandNum == gf.levelIntermediate) {
                    System.out.println("change to intermediate");
                    gf.numOfVaccines = 7;
                    gf.gameLevel = gf.levelIntermediate;
                    gf.mc.setDefaultValue(gf.tileFrame.getStartPoints(gf.levelIntermediate));
                    gf.gameState = gf.titleState;
                }else if(gf.commandNum == gf.levelChallenge) {
                    System.out.println("change to challenge");
                    gf.numOfVaccines = 10;
                    gf.gameLevel = gf.levelChallenge;
                    gf.mc.setDefaultValue(gf.tileFrame.getStartPoints(gf.levelChallenge));
                    gf.gameState = gf.titleState;
                }else {
                    System.out.println("invalid commandNum");
                    gf.gameLevel = gf.levelEasy;
                    gf.numOfVaccines = 5;
                    gf.gameState = gf.changeLevelState;
                }
                gf.commandNum = 0;
//                  gf.gameState = gf.titleState;
            }
            } else if(gf.gameState == gf.playState) { // key input during play state
```

b. inside inputKey() class here ^, this chunk can be put inside a function that takes commandNum as input and sets the variables

c. also for further improvement you can also make a dictionary that has all these variables, for example if gf.levelEasy value is 1 then

   i. {1:
      {'numOfVaccine': 5,
      'gameLevel' : 1,....and so on}} -- this way you don't have to add so many if statements

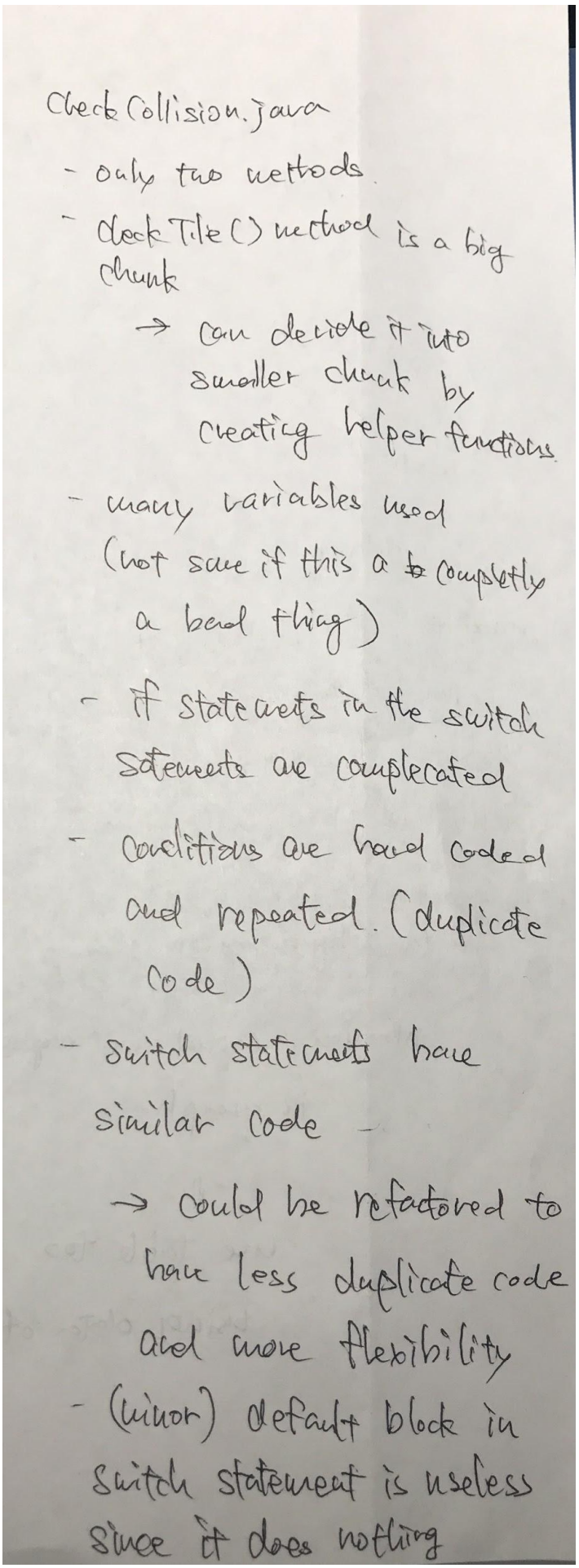
3. poorly structured code

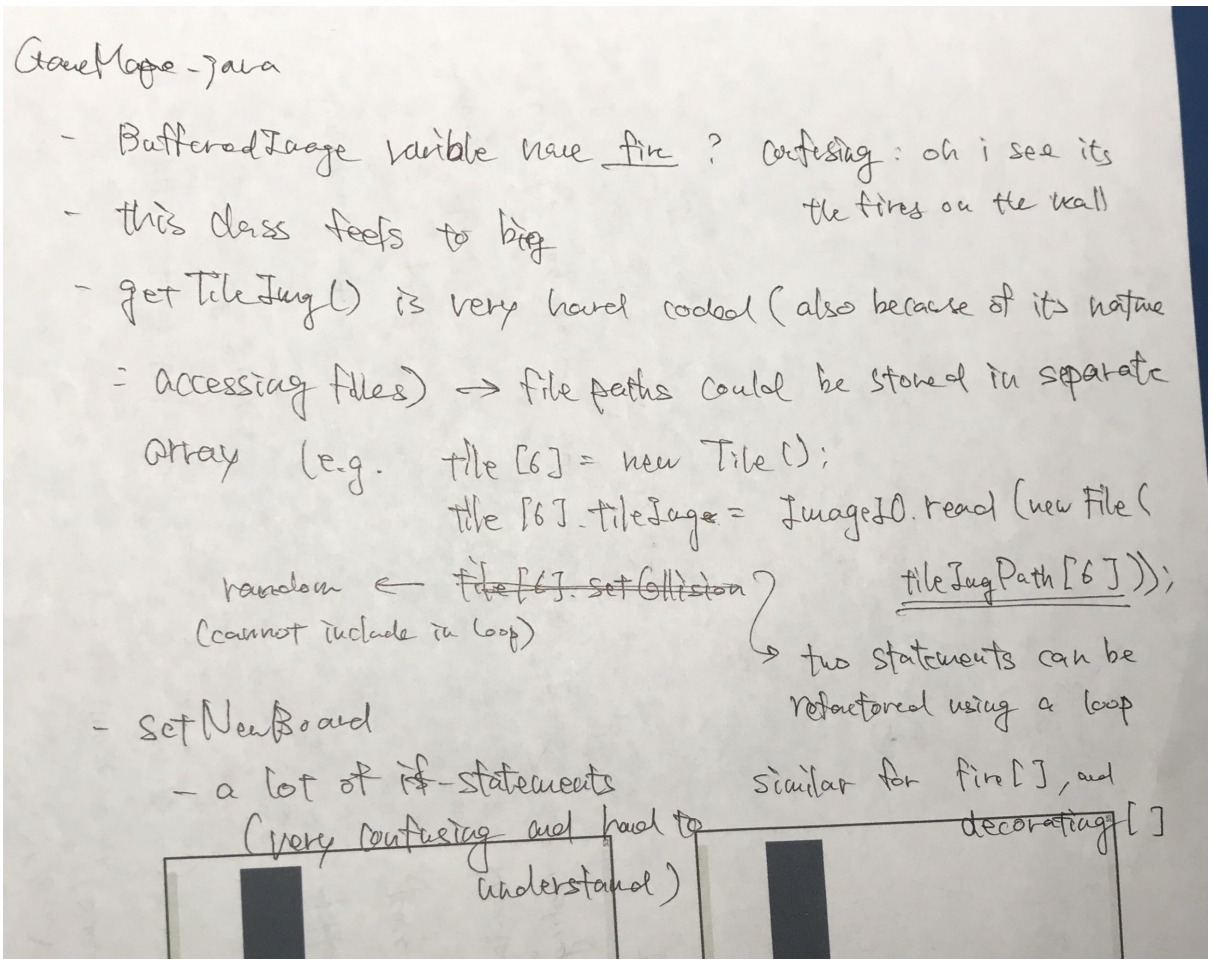
```
203         setUpScreen();
204         setStartPoint(100,100);
205
206         bgImage = new ImageIcon("src/main/java/picture/GUI_image/titleScreenBg.jpg").getImage();
207         bgTitleScreen = new ImageIcon("src/main/java/picture/GUI_image/title_screen_bg.png").getImage();
208         bgChangeLevelScreen = new ImageIcon("src/main/java/picture/GUI_image/change_level_screen_bg.png").getImage();
209         overlayImage = new ImageIcon("src/main/java/picture/GUI_image/overlay_instructions.png").getImage();
210         titleScreenOverlay = new ImageIcon("src/main/java/picture/GUI_image/title_screen_overlay.png").getImage();
211         endScreenOverlay = new ImageIcon("src/main/java/picture/GUI_image/end_screen_overlay.png").getImage();
212 //        winEndScreenOverlay = new ImageIcon("src/main/java/picture/GUI_image/win_end_screen_overlay.png").getImage();
213 //        failEndScreenOverlay = new ImageIcon("src/main/java/picture/GUI_image/fail_end_screen_overlay.png").getImage();
214         gameWinBg = new ImageIcon("src/main/java/picture/GUI_image/escape_success_bg.jpg").getImage();
215         gameFailBg = new ImageIcon("src/main/java/picture/GUI_image/escape_fail_bg.jpg").getImage();
216         cursor = new ImageIcon("src/main/java/picture/GUI_image/redHand_icon.png").getImage();
217
218         // get images for buttons in end screen
219         retryButtonRegular = new ImageIcon("src/main/java/picture/GUI_image/RetryButton_noLight.png").getImage();
220         retryButtonLight = new ImageIcon("src/main/java/picture/GUI_image/RetryButton_light.png").getImage();
221         returnButtonRegular = new ImageIcon("src/main/java/picture/GUI_image/ReturnButton_noLight.png").getImage();
222         returnButtonLight = new ImageIcon("src/main/java/picture/GUI_image/ReturnButton_light.png").getImage();
223
224
225         // get images for tutorial screen
226         tutorialPage = new ImageIcon("src/main/java/picture/GUI_image/tutorial/tutorial_sample.png").getImage();
227         tutorialPage1 = new ImageIcon("src/main/java/picture/GUI_image/tutorial/tutorial1_sample.png").getImage();
228         tutorialPage2 = new ImageIcon("src/main/java/picture/GUI_image/tutorial/tutorial2_sample.png").getImage();
229         tutorialPage3 = new ImageIcon("src/main/java/picture/GUI_image/tutorial/tutorial3_sample.png").getImage();
230
231         // get images for narration screen
232         narrationPage1 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story1.png").getImage();
233         narrationPage2 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story2.png").getImage();
234         narrationPage3 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story3.png").getImage();
235         narrationPage4 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story4.png").getImage();
236         narrationPage5 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story5.png").getImage();
237         narrationPage6 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story6.png").getImage();
238         narrationPage7 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story7.png").getImage();
239         narrationPage8 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story8.png").getImage();
240         narrationPage9 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story9.png").getImage();
241         narrationPage10 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story10.png").getImage();
242         narrationPage11 = new ImageIcon("src/main/java/picture/GUI_image/background_story/background_story11.png").getImage();
243     }
244
```

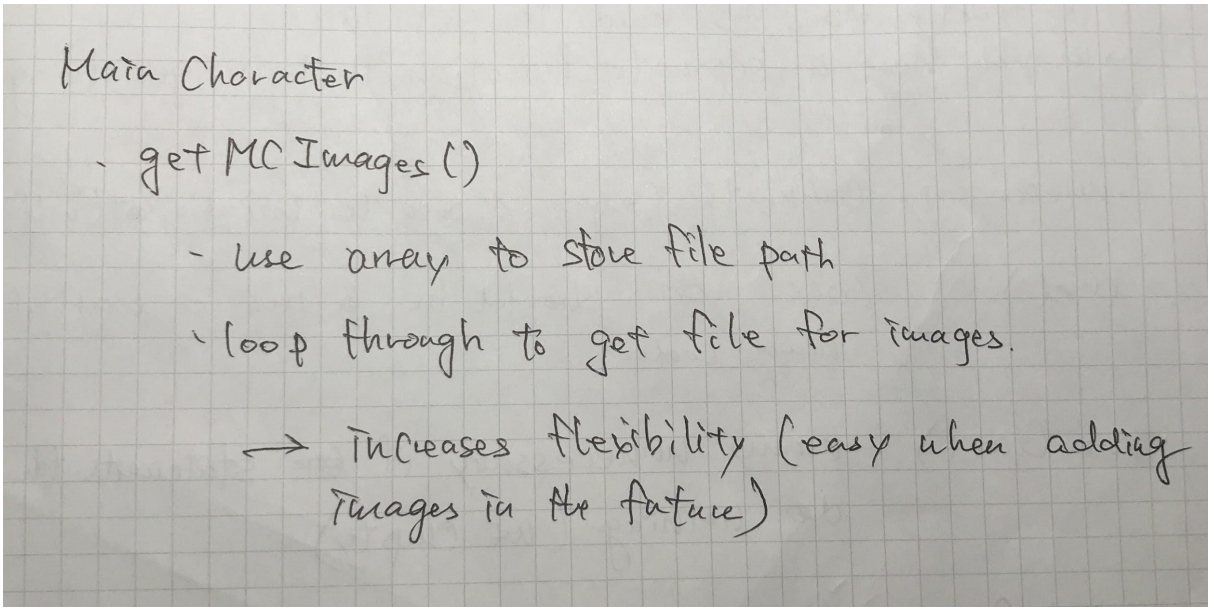
a.

b. its not good practice to directly initialize image import inside constructor, you can put these in a function and call them in gameframe constructor

4.


CheckCollision.java
- only two methods
- checkTile() method is a big chunk
→ can decide it into smaller chunk by creating helper functions
- many variables used (not sure if this a ~~to~~ completly a bad thing)
- if statements in the switch statements are complecated
- conditions are hard coded and repeated. (duplicate code)
- switch statements have similar code
→ could be refactored to have less duplicate code and more flexibility
- (minor) default block in switch statement is useless since it does nothing

5. 

6. 

7. seven

Some ideas for things to look for include:

- bad/confusing variable names
- methods that are too long and that could benefit from being refactored

- classes that are too large and/or try to do too much
- low cohesion
- high coupling
- long list of method parameters
- lack of documentation
- poorly structured code
- confusing class hierarchy
- badly structured project (i.e., file setup)
- dead code
- code duplication
- unused or useless variables
- unnecessary use of unsafe or unsound constructs
- data clumps
- unjustified use of primitives
- unnecessary if/else or switch/case statements

Solutions we used to address the detected smells

1. one
2. two
3. three
4. four
5. five (min)
6. six
7. seven

Performed refactoring

1. one
2. two
3. checkCollision DOne
4. five (min)
5. six
6. seven

Instruction given for assignment3:

You will work in groups of two. Pair up with one of your team members. You will choose a part of the code that the two of you have implemented, over which you will perform a manual code review. Identify at least five code smalls in code written by one of the two people performing the code review. Refactor the identified smells to improve your design and code quality. Rerun your tests after each change to ensure you are preserving the behavior of your program. Make sure to continually commit and push the refactored code. In your report, document the smells you have identified, and your solutions for addressing the detected smells and the performed refactoring. Point to the relevant code and commits from your report when explaining the refactorings.

Assignment 3: Code Review Report

Introduction:
Initially in this assignment, we had discussions where we broke down our game code into chunks that were implemented by us on the majority scale. Each of us looked over the sections that were developed by the other to recognize any bad smells within our code. We used the guidelines provided in the Assignment Manual to help us recognize the bad smells. When overviewing the entire code, we had found the code to follow high cohesion, and low coupling for the majority of our classes and methods. On the contrary, we found the code to include several sections of poorly structured code, and significant lack of documentation. The details of several other bad smells including the ones mentioned above are discussed below.

Detected Bad Smells and their Refactoring:
A very common bad smell within our code was **poor documentation and code structure**. This included methods and classes that were significantly long with minimal documentation on the process. Specifically the checkCollision class included one large method to perform all the collision detection. Upon further observation, it was notable that the method was unnecessary long due to duplicate/similar code in several conditions of the if statements. With a slight change in some variable placements, this could be refactored by adding a helper function to handle duplicate sections of the code. This helper function was then called inside each if statement. The detailed refactored code can be seen in commit `b295f2b5` which was applied over the checkCollision class within the checkTile method. The helper function added was named setTileCollisions.

Similar bad smell was detected within the inputKey class inside the keyPressed() method which handled all the cases and scenarios for player inputs. To account for all input cases in different screens (Main Screen, Tutorial Screen, Settings Screen) all solutions were coupled into one large method. This led to multiple sections of code duplication, and in some cases dead code. As per commit `00edd632`, we implemented additional performActionWithEnter() methods with different input parameters to handle player inputs in each screen of the game. This was then called within the if statements which made the code more readable and less clustered. This commit also solved the lack of documentation issues inside the inputKey class. We added additional comments regarding each scenario and their functionalities.

We found the GameFrame class to have very low coupling, as this class was used as an interface to integrate all other major classes. This resulted in the class being significantly long and not legible. After further observation we found the class could use helper classes to handle some functionalities implemented in GameFrame. Inside the same commit, `00edd632` along with `9541a6df`, we added a Command class which worked with

returning and setting the game settings attributes such as the chosen level and its relative Game objects. Additionally, we added the TitleScreenCommand, EndScreenCommand and ChangeLevelCommand classes which extend the Command class. The EndSceenCommand class handles functionalities of the screen the player sees once the game ends with either a winning or a losing condition. There are two command options retry and return which are initialized in the constructor by adding it to the command list(commandOption). The ChangeLevelCommand class implements functionalities within the screen which the player sees when changing the game level. After adding these classes, we were able to integrate them inside the GameFrame class and eliminate clustered code. This made the GameFrame class manageable in any cases of debugging in the future.
This change also reduced a big amount of duplicate code or similar structure code in inputKey class. Since the command number is controlled internally in the class using incCommand() and decCommand() methods, the Client user inputKey does not have to deal with the internal structure of the command. As a result, we managed to significantly reduce the amount of lines in the keyPressed() method of inputKey class.

Many parts of the program (mainly GameFrame and inputKey class) contained duplicate code which was dealing with the game state. The GameFrame class also contained 19 game states and its variables. In order to improve the code structure, we added the GameState class which deals with the game state. This class also controls the state change with the methods such as toNAME_OF_STATEState (e.g. toTItleState, toPlayState). Also, instead of controlling each page in the tutorial state and the narration stage we used a single variable ‘page’ to keep track of which screen is displayed in the state(tutorial/narration) since we know the order is always from tutorial state to narration state and to play state. Methods: nextPage, prevPage, skipTutorial, and skipNarration deal with the internal structure such as checking the boundary of the page number. This functionality was duplicated in many parts of inputKey’s keyPressed method. Therefore, gathering it in one place and having one class being responsible for it reduced a lot of duplicate code as well as many unnecessary if-else statements.

The GameFrame class also included a constructor which was significantly long due to many declarations of image import. We found this to be poorly structured and an use of unsafe constructs. We were able to easily solve this by implementing methods which handled the image imports. This was then called within the constructor to ensure observance of the same features and GUI within the game. The detailed refactored code can be seen inside commit `f9b71883`.