War of Robotcraft

Construction Experience Document

Team: A3

Team Members:

Name	NSID	Student ID
Fu, Chen	chf354	11183491
He, Jiahuan	jih889	11183346
Wang, Shisong	shw940	11157916
Xie, Ruida	rux793	11194258
Yang, Chen	chy202	11183550

Date: Nov 24, 2016

Document History Log

Version Number	**Description of Changes**	**Approved Date**
0.1	First Draft	2016-11-24
0.2	Add pair programming.	2016-11-25
0.3	Add code review summary.	2016-11-25
0.4	Add Amendment.	2016-11-25
0.5	Modified pair programming.	2016-11-26
0.6	Modified code review summary.	2016-11-27
0.7	Modified Amendment.	2016-11-27
0.8	Final edition.	2016-11-27

1. Pair Programming

1.1. Chen Fu and Chen Yang

What we had done is the development of the RobotLabel class which inherits from JLabel creating a label with the image of a robot. At first, we talked about how to develop this class for about ten minutes. Since we did not design this class in the design phase, we analyzed the fields and methods of the class and how to be invoked in other classes, then we designed methods one by one including return type, parameters passed in, and the basic logic. After that, we started to implement this class. Firstly, Chen Yang coded the fields, constructor, and a helper function for the constructor, and Chen Fu sticked her eyes on the code to check if there was something incorrect. Fifteen minutes later, we exchanged the role that Chen Fu coded the rest part of the class and Chen Yang monitored what was written until this class done. Actually, it was the first time for both of us to do the pair programming, and we find it really efficient and productive that it only took us forty minutes to discuss, develop, test and troubleshoot. More importantly, the most efficient part we discovered is one person coding and another navigating because the person monitoring on the code provides many useful reminders so that it reduces a lot of time to revise the code.

1.2. Chen Fu and Jiahuan He

In this pair programming session, we develop the GameBoardView class. This view is really important part of this project, and the most difficult thing is how to implement the hexagon game board. First, we want to use button to present every small hexagon, and we found that if we don't customize button, we can not change the button shape, so the control of the event response area can not set precisely. Then we figured out many ways, and decide to use a easy way which is using labels. However, layout of swing is another difficulty, Jiahuan played the ability to search for information, and he found many examples for us to learn. Finally, we implemented our design perfectly. After this session, we have a better understanding of the characteristics of swing, and learned to use different methods to solve the problem. Also, we have good experience in pair work.

1.3. Chen Fu and Ruida Xie

Chen Fu and Ruida Xie work in pairs to implement GameStartView. This class will provide the main interface of this game, users can select start, garage and exit on this interface. This interface is relatively easy to implement because it only has three buttons and we had drawn the background picture in our requirement document. Ruida is the pilot at beginning, then Chen added some effects to these buttons to make them more realistic. Chen is quite familiar with interface design, so we can breeze through this task.

1.4. Chen Fu and Shisong Wang

We worked together to implement the SetGameModeView class. To make the user interface consistent with our design documentation, we learned about the use of swing and then decided on the best implementation of each section. For example, in the SetGameModeView, we used the combination of labels and radio buttons to solve radio button can not customize the picture. Another, we figured out how to implement event response among the radio button, combo box and button in this view.After this pair programming session, we found that this is a effective and efficient way to develop the project. Since we both aren't familiar with properties of controls in swing, we tried many ways to implement design. During this process, we leaned a lot of new skills and avoid wasting time in wrong direction.

1.5. Chen Yang and Jiahuan He

This is the pair programming sessions of Jiahuan He and Chen Yang. During these sessions we progressed the project by developing the hashmap mechanisms to store the coordinates and players information after analyzing the time complexity of the software and considering the ease of use of the data type's interfaces, and we also figured out the helper methods in some model classes to generate the key pairs of directions and absolute coordinates in order to support controller and view class. We've found a lot of improvements from the experience, including the skills of transferring the information of problems and discussing solutions. After these sessions, we ended up with satisfying development experience and outcome.

1.6. Chen Yang and Ruida Xie

Chen and Ruida paired together to implement the shooting action for a robot. We completed three methods: the shooting action method in the robot class, the update method in the game class, and the shooting helper method in the controller class. We shifted our roles for three times, and each time lasting for about twenty minutes was for implementing a method. When a person was writing the code, the other was looking at the code and checking if the code was correct. If the person who navigating the code found something incorrect or imperfect, he reminded the coding person immediately and the code was revised. By this pair programming, we saved a lot of time to implement this part because we discovered the fault as soon as possible by the navigating person. More importantly, Since we paired together, we had a lot of time to discuss then learned a lot from each other not only the coding skill but also the perspectives to think and work out a problem. Therefore, we both believe it is an productive way in coding, and an effective way to learn.

1.7. Chen Yang and Shisong Wang

In the progress of pair programming of Chen Yang and Shisong Wang, Chen played the pilot role and Shisong played the co-pilot role. We wrote code in controller class and we tried to make the whole game become alive. For doing that, we need to receive user input and handle it by setting data in model classes. Also, we need to update the UI to make user can observe the changes. When we write the logical part of controller which update the data in model classes, it is important to us to be familiar with the model classes. As Chen took part in writing model classes, he could easily figure out how to call the methods in model class to update the data. When we write the code for update UI, as Shisong participated in writing UI, Shisong can assist Chen to figure out which methods in UI classes should be called to update specific information. After the pair programming experience, we realized that two people write code together can accelerates the rate of progress especially when two people are familiar with different part of the project and we need to associate those two parts.

1.8. Jiahuan He and Ruida Xie

Jiahuan and Ruida work in pairs to implement the RotateImage class. This class is served for robot turning, each time a robot turns to a new direction, the icon of this robot will rotate correspondingly. Thus, users can visually see the robot facing direction. At the first stage, Jiahuan is pilot and Ruida is co-pilot. Since both of us lack experience in solving similar problems, we try to search some helpful information on the internet at first. Then we decide to import Graphics2D to solve this problem. We also encountered difficulties reading robot icon, Jiahuan used read file to read robot icon but it did not work. Ruida suggested use ImageIO instead of read file, then we solve this problem perfectly. We think pair programming is an effective way to write code, we can exchange our ideas.

1.9. Jiahuan He and Shisong Wang

In these pair programming sessions, Jiahuan He and Shisong Wang worked together on implementing the design of turn action of each robot and writing the method of run the plays of the game. In order to figure out how to make the robots able to perform turning action, we reviewed the related parts in model and view class. From these pair programming sessions, we found the significance of reviewing all related parts when dealing with interactions between different system components. In addition, we did some debug works on updating war mist and we found that pair programming is not only a good way to write code but also an efficient approach to debug existing code.

1.10. Ruida Xie and Shisong Wang

We implemented the operations of human player such as pressing key to move, shoot, turn, and surrender. As we do not familiar with the key pressed listener and the way to handle the events enough, we were blocked by adding a listener and making it work. Then we tried to writing the logical event-handling code first by one person while the other person reading the document about key pressed event handling. When the person who were writing the event-handling code finish the logical part, the person who were reading the document took over and added the key pressed event listener and make the logical part work. From the pair programming experience, we found that it speeds up our coding progress because if there is only one person doing this work, it is too easy to be stuck. However, in a pair programming process, we could try to solve a problem while the other person is keeping coding, which would massively increase the whole coding process.

2. Code Review

2.1. Reviewed Code

The code we select to review is the Game class. That's because the lines of code of this part fit the length requirement of 200-400 lines. In addition, they are very critical for the game system to run as the game required because this class has composition relations with many other classes, and this class interacts with controller class very often. Also, we find some bugs when we want to make use of some methods in this class, which affect the performance of interactions between the robots and the system, such as the robots are not displayed correctly when some certain inputs are given and the war mist is not updated correctly on the game board view occasionally. Thus, we pick up the class to review and try to solve the bugs.

2.2. Code Review Activities

After selecting the code, we conduct the code review session by carrying out the following activities. First, we send out the information of the part which we will review to the team. The information contains the introduction of the code as well as the data structure, such as hashmap and linked list, and the classes description for the purposes of helping the team members get familiar with the content of the review. Before the code review session, each team member has also read the classes which have interactions with the game class. So that we can read the code thoroughly to do some preparations. During the review session, we look at the code together and communicate with each other to look for problems. To avoid overlooking potential flaws, we compose a checklist in advance, and we examine the source code following the checklist. Finally, we discuss the issues found in the review session and try to solve it after code the code review session. Beyond this part of code, we also discuss some possible improvements for the further coding sessions.

2.3. Code Review Content

2.3.1. Good Points

After reading through the piece of code, we have found some good points that we can keep doing. First, each method have a piece of block comments. The comments are intended to describe the purposes of each method, and they indicates some important information such as return type and parameters. Also, the class's methods and variables follows the naming conventions such as camel case and Hungarian notation basically. Modular method is another good point of this class. Each method in this class has distinct functionality, so that they work together coherently with weak coupling. For example, there are two methods that deal with shooting actions. One is responsible for the situation that the robot takes the damage and still alive, and another is responsible for the situation that the robot is dead after taking the damage. So that each of them does very specific tasks and not interfere each other.

2.3.2. Bad Points

Besides the good points, we also found some problems during the code review session. One of them is that we found some places using long redundant code and not considering writing a helper function to deal with multiple functions. Another one is that we forgot to write sufficient inline comments. Also, there are some pieces of code which have been commented out, but there is no explanation of them, which looks messy and could be confusing in the future. In addition, there are warnings with which have not been dealt yet. For the coding style, although we followed the coding style basically, we missed some details of it, such as some spacing issues and indentation.

2.3.3. Discussion

As reviewing the piece of code line by line, we find some of them need to be discussed further. The followings are what we discussed:

2. 3. 3. 1. Coding Style

We have to stick to the coding style that we decide. Since each team member has his or her own coding style to follow in the past coding experience, we are not strict in accordance with our style sometimes. We all believe that this will go worse in the future if we do not pay more attention to the coding style, and this would lead us to read code more difficult even confuse by the code. To avoid this situation happening in the future, we discussed some ways, and three ways are probably to be taken. The first is the navigating person in the pair programming pay attention to the coding style, and reminder of the coder once discover. Secondly, when other team members reading code and discovering something not in our coding style, change the coding style to ours. The last one is each of us is supposed to read what have written before committing to git, and this way is not only suit for the coding style but also for comments, variable name etc..

2.3.3.2. Comments

The comments must be written properly and concisely. After reviewing our code, we find that we have written java doc before each method but some are not in proper way and some are wordy. Also, we find there are hardly any inline comments, and it is time consuming for other team members to read the code and take a lot of time to revise the code if some bugs found. Even sometimes, the coder is hard to figure out the code clearly when looking back. Thus, we talked about two points to improve our comments in the future. Firstly, we talked about the template to follow for the Java doc to avoid wordy and confusing, which is this method is to do something or reach a specific goal or create something. This comments should be finished in one line, and must be done in two lines if this method is complicated. Also, this template is applicable to the notations in Java doc. Secondly, we talks about the inline comments. We must write the inline comments when pair programming. As we talked above that read the code after developing a method, we must supplement inline comments and the navigating person should check if the comments make sense.

2.3.3.3. Duplicated code

We should make good design to avoid the duplicated code in the future. When we reviewed the method goNextPlayer in Game class, we found duplicated code in each case branch. All of us think this duplicated code does not follow the structural requirement of software engineering, which could lead to low code reusing and inefficient development. So we talk more on how this could happen and how to avoid in the future. We all believe that this happens due to the imperfect design. Since the design document concerns about the overall system and the global structure, there is not much detail for each method about how to implement. We should discuss and think more in detail when pair programming for each method in the future. If we find the duplicated code is going to occur, this duplicated code is supposed to be developed in a separated method and make it invokable.

2.3.3.4. Multiple functions in one method

We should implement each method to achieve one goal. As going through our code in the game class, we find the method named updateGame is a super large method to realize the whole game updated when some actions performed by robots. It takes us a lot of time to discuss on how to separate the method to different methods for different actions. As a result, we all think we should separate this method into two method which are updateGameShootDamaged and updateGameMove. Thus, this helps to enhance the code execution efficiency. Also, we talked about why this happened and how to avoid as well as something how to enhance the code execution efficiency. We all believe this is because we follow our design document to develop methods and we focus more on the whole system and structure in the design phase. Therefore, we decide to discuss in detail for each method before writing code when doing pair programming. If method is going to perform more than one functions, we should separate it into small one. Also, we should call other teams to meet to talk more on this separation if this change could affect other methods.

2.3.4. Improvement

We focus more on inline comments from now on. Because we realize that if there are not sufficient comments, it is hard to recall the complex logic of some methods if we need to modify some code in the future. We are also more aware of the importance of the clarity of the source code since we have emphasized it during the code review session. And we have made an agreement that everyone should be following the code convention more carefully when writing code since if everyone pays attention on that from now on, there would not be a lot of extra tasks to correct the code format later. After analyzing some duplicated examples in the game class, we improved some redundant code. We learned from the discussion that we have to avoid repetitive code, and we should use modular code and make use of existing code more often. So that the development of the project would be more efficient and the code would be more readable. The functionalities would be allocated to distinct methods more specifically, in order to make the logic and the code structure clearer.

2.4. What we have learned from the code review

It takes all of us two hours to do the code review, it not only helps us to revise our code but also we learn a lot. Firstly, we obtain deep understanding on the importance of coding style and comments. That's because the coder spend much time in explaining the code to us. Secondly, we know more on duplicated code and the cons of it. Also, we are clear on how to revise it. At last, we learned how to separate a large method into separated small method and make them invokable.

3. Amendment

3.1. Model Package

3.1.1. Coordinate Class

We add getRing(), getRange() and getNewCoordinate() methods. These three methods are both helper funcitions dealing with robot move, turn, shoot and mist updating.

3.1.2. Game Class

We add a field called alivePlayerNumber, this field stores the amount of alive players. It can make the system determine who is the winner. We also split updateGame() method into 3 methods, updateGameMove(), updateGameShootDamage() dand updateGameShootDead(). We had trouble in implementing updateGame() method in the Game class according to our design document, because we have not cosidered this method carefully enough. The updateGame() method is a larger method will handle many different situations including robot move, shoot, turn, damaged, do nothing, do nothing but move, surrender, etc. It also transmitted a lot of data and call a lot of methods from other class. We divided it into three parts, each part takes different responsibilities. The advantage of doing this is reduce code coupling and facilitate our team to debug.

3.1.3. Map Class

We use hashmap to store the coordinates and players' information instead of using list of pair. In our design document, we did not consider the situation where more than one robot stops in one grid, so we chose list of pair. Using hashmap can reduce time complexity. It is convenient for other methods to use this result.

3.1.4. Player Class

We added a currentRobot field to store the robot for playing because it can help us to simplify the process to get a object of current playing robot. And for an easy access of the view range of all robots in a team, we added a field called viewRangeList to store the view range of all robots in same team and a method called updateViewRange to update the field. Also, we added a surrender method for a player to give up, which was not considered as part of our design.

3.1.5. Robot Class

As we did not think into update the visible range deeply enough, we found that we cannot get visible range easily enough. Therefore, we add a viewRangeList field in Robot class to store the view range of a robot and to be updated when the robot is moved or died. And in the process of coding, we realized that we should reset the movingPoint, hasShot, and hasPlayed fields, we add a new method called resetStatus to reset those fields when a play starts. Also, we add relativeDirectionToCoordinate method for the future UI display purpose.

3.2. View Package

3.2.1. GameBoardView Class

We added updateTimerNumber, updateMist, updateCurrentPlayer, updateCurrentRobot, updateOperationState, updatePlayerDeath, updateRobotLocation, updateRobotTurned, updateRobotDamaged, and updateRobotDestruction methods. In our previous design, those methods are included in controller class. When we write the code, we realized that it is hard to implement updating UI if we put those methods in view. Thus, we move them from controller to view. Then the game data would be passed from controller to view by calling those methods in GameBoardView Class. Also, to make view less coupled with other part of the project, we changed the type of data passed in view to basic types.

3.3. Controller Package

3.3.1. Controller Class

We add four fields, gameBoardViewTimer, operationMode, shootDistance and shootTarget. The gameBoardViewTimer is the system built-in timer, which controls timing of each turn. In our design document we only designed a timer method in the Controller class, however, it is not enough. Thus, we add the timer field. The reason we add the other three fields is similar. We decide to store them separately to increase the readability of the code. Now other methods can use these fields' getter method as parameters instead of using a long method call.For methods, we remove updateRobotPosition(), updateRobotHealth() and updatePlayerDeath() methods.

We move them to GameBoardView class. It will reduce code coupling between GameBoardView and Controller and make data transmission more efficient from Controller to GameBoardView.