International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.3, Issue.2, March-April. 2013 pp-1129-1134 ISSN: 2249-6645

The Design and Implementation of an Android Game: Foxes and Chickens

Justin R. Martinez, [1] Wenbin Luo[2]
[12]*Engineering Department, St. Mary's University, San Antonio, United States*

Abstract: *In this paper, we present the design, implementation, and testing of an Android game, called Foxes and Chickens. It was developed for mobile devices with the Android mobile operating system. The Foxes and Chickens game itself has origins in South America, where it is very popular in some countries. However, to the best of our knowledge, no digital version of the game was available for mass consumption. Our research seeks to remedy this vacancy with the goal of adding values and enjoyment to users of the Android mobile devices. After going through strict software engineering processes of specification, design, coding, and testing, we successfully developed the game, as shown at the following link: http://www.youtube.com/watch?v=xeXYiR3Qwio. The game Foxes and Chickens is available for download from the Google play for anyone with an Android powered device. It has been tested for compatibility on the following mobile devices: T-Mobile's G1, HTC My Touch, and Verizon's Motorola Droid. It should run on other compatible Android handsets as well.*

Keywords: *Android development, foxes and chickens, Java, mobile games, software development*

I. INTRODUCTION

The Foxes and Chickens game is developed for mobile devices with an Android operating system, which was designed as an operating system for mobile devices by Google and the Open Handset Alliance based on the Linux kernel [1]. The Foxes and Chickens game is a turn-based strategy game that has similarities to Checkers and Chess (rules of the game can be found in Appendix). The game has two game modes (single player and two players) and three levels of difficulty (easy, intermediate, and hard). The object of the game is to move nine of the initial twenty chickens on the game board to safety, while the foxes attempt to prevent this by killing chickens within reach.

At the time of creation, game development on the Android platform was still in its early stage and few projects existed where the development process and sources were available for others to utilize. This research sought to remedy this vacancy by both documenting the creation and providing source code for others to learn from.

1.1 Product Features

A high level list of the major features of our developed game is as follows.

- Two operational modes: single player and two players.
- Three tiers of difficulty: easy, intermediate, and hard
- Invalid movements are not allowed.
- In-game rule-set, tutorial mode where invalid moves are explained.
- Non-Player Character (NPC) movement accomplished in fixed time.
- Intuitive and linear interface.
- Smooth animation system.
- Bright, vibrant colors.

These represent the major categories of the requirements and will be explained in more details in the remainder of this paper.

1.2 User Classes and Characteristics

There is one class of user for the game: the player class. The player can be either male or female, most likely technically savvy (due to the target platform), and can be classified as a casual gamer. Casual gamers tend to enjoy simple, yet dynamic games that are easy to understand, frequently reward the player, are short in duration (as opposed to games where characters must be developed and nurtured), and have high re-playability by not becoming boring or repetitive.

1.3 Online Help

A help screen will be provided to assist in understanding the rule-set for the game. No other documentation will be provided to the user. The linear and clean structure of the user interface makes additional help unnecessary. One of the help screens is shown in Fig. 4 of section III: user interfaces. Help screens are intended to give the player an overview of how the game is played.

II. SYSTEM ANALYSIS AND DESIGN

The system analysis of the proposed Foxes and Chickens game involved the formulation of use cases, sequence diagrams, and state diagrams of the game. To properly understand and explicate out the game's flow, we developed a number of use cases. All of these techniques allowed us to better articulate the requirements of the system in terms of objects, classes, attributes, relationships, scenarios, and actors. A sample use case is shown below.

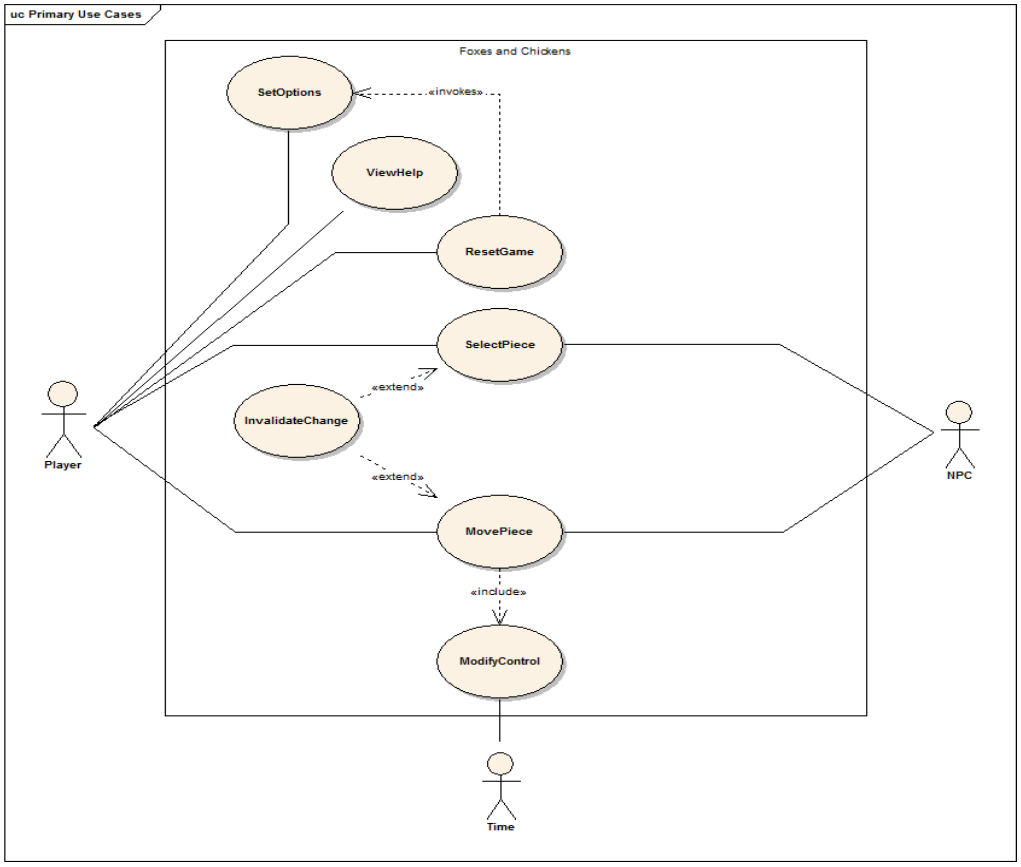

Figure 1. Use case

Sequence diagrams are the interaction diagrams that show how processes operate with one another and in what order. The aforementioned use case was captured in the following sequence diagram which we developed for the *Foxes and Chickens* game. In this use case, the player launches the game or has selected to reset the game while playing. If the player has launched a new game the default values are used for the selection available to them, while a reset game will use the setting from the previous play. After accepting the defaults or correcting them to the player's desire, the game is launched.

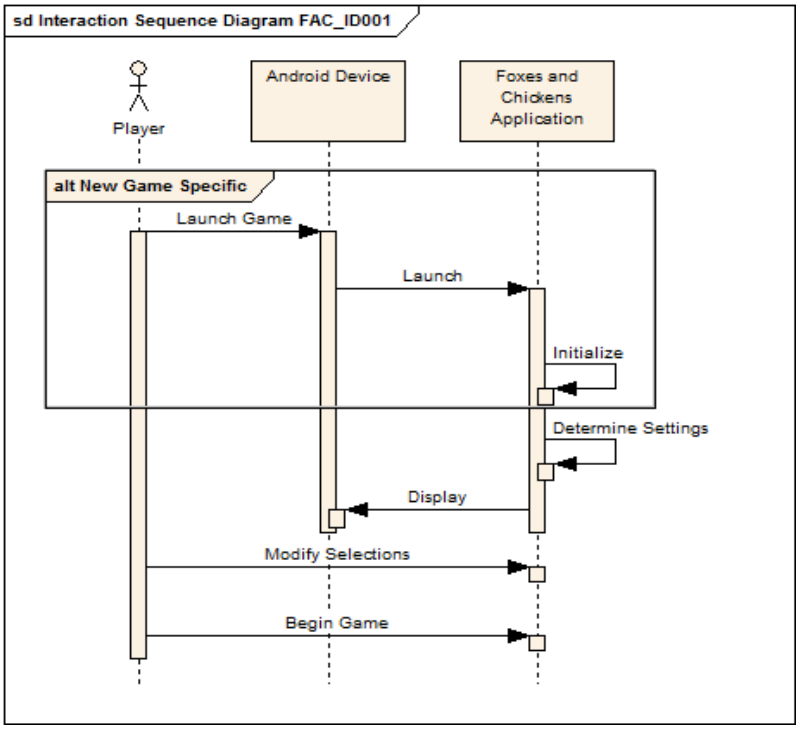

Figure 2. Sequence diagram

At its most basic, the game functions as a simple state machine. As the player interacts with the system, control flows between discreet points, each with separate logic. When the game is started, the user is prompted to select the number of players, dynamic help and whether to enable audio. If the number of players is "one", the user is then asked about the difficulty level. After that, the game begins. After the user selects a piece to move and selects a destination, the move is validated. In the case of the NPC mode, validation is skipped as it is an integral part of the artificial intelligent process in the system. The system then checks the game for a *game over* state. If the game is over, the user is prompted to reset. Otherwise, the active player is toggled and the process cycles.

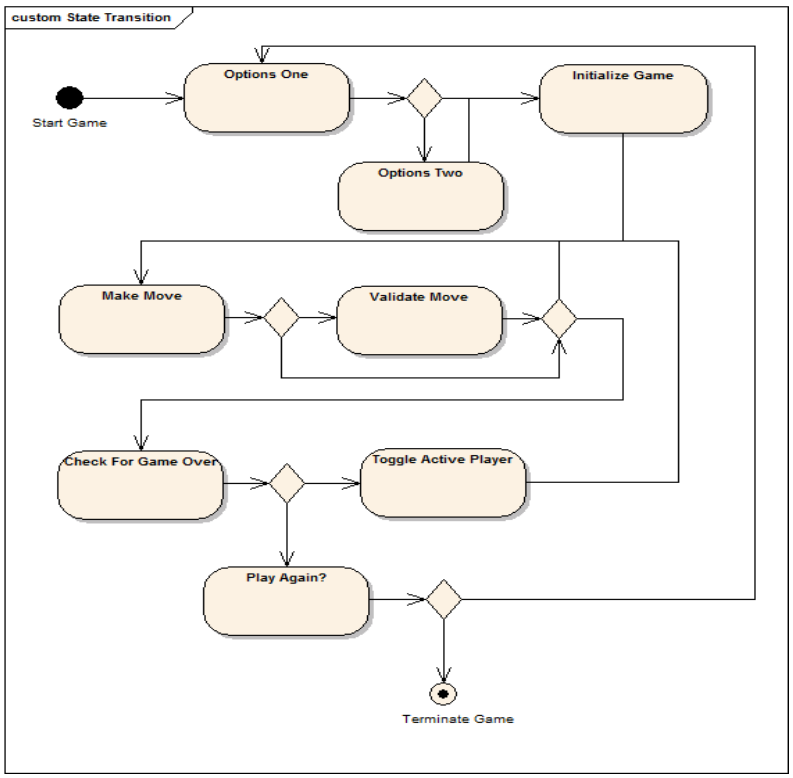

Figure 3. State map

III. USER INTERFACES

Usability and portability are the most important quality attributes applicable to this research project. Since the *Foxes and Chickens* game is targeted toward casual gamers, the interface and mechanics of game play are required to be simple and straight forward. This section provides several sample screen images representing the general "look and feel" of the interface and a visual representation of applicable system features and game states.

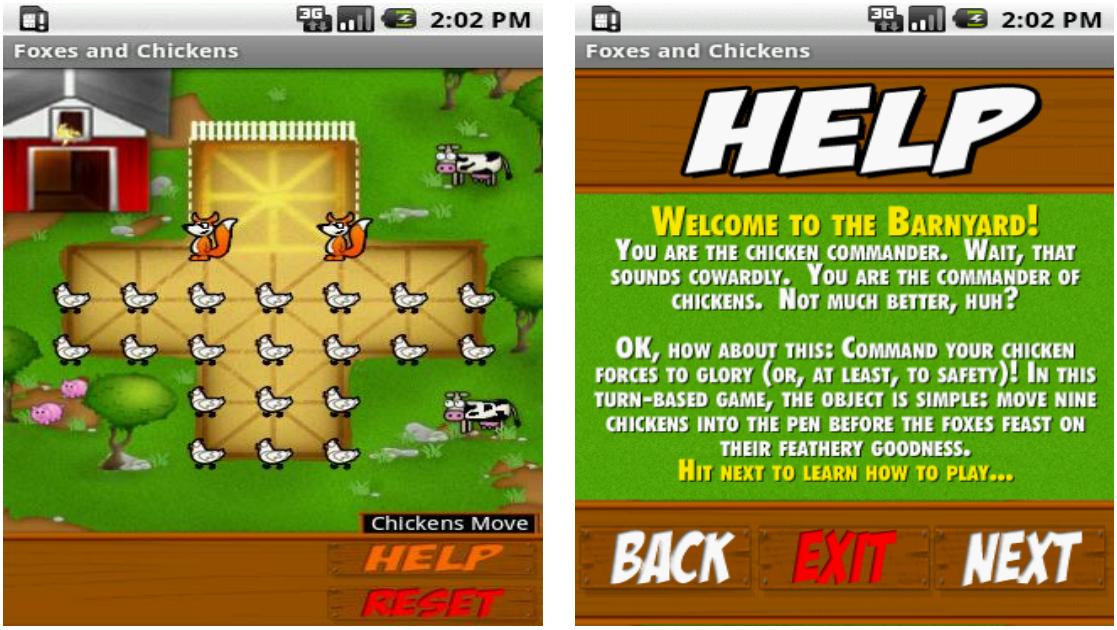

Figure 4. Sample screen images

Upon initial launch, the game enters *SelectPlayersState*, where the user notifies the system of the number of players. In the sample images presented, the user would have selected "One". This selection transitions to *Select Dificulty State* where the user notifies the system of the difficulty level at which he could like to play. After *Select Difficulty State*, the system has enough information to initialize the game and allow game play to begin. *Game play State* is not a single state but an aggregation of several states. Upon completion of game play, the system transitions to the *End Game State*, where the user is notified of his success or failure.

When the user options to be notified of invalid movements, he will see a text message appear in a notification area after an invalid move attempt.

IV. IMPLEMENTATION AND TESTING

The software development utilized the Eclipse Integrated Development Environment (IDE) with the Android Software Development Kit (SDK) that is cross platform among Windows, Linux, or Macintosh. In the case of this research project, the following configuration was used: Windows Vista SP2, Java 1.6, Eclipse 3.5, and Android SDK 1.5r3. Testing occurred on a T-Mobile G1 and HTC My Touch running the Android OS, versions 1.5 and 1.6 and on a Motorola Droid running the Android OS version 2.0.

The figure below shows the deployment model for the Foxes and Chickens game. As stated previously, required is a T-Mobile G1 Handset running the Android OS version 1.5. When packaged within Eclipse, the resultant file is of type *.apk, which is an encrypted and compressed java package. When the game is installed on a target device, the application icon is extracted and yet the application itself remains in the apk form.

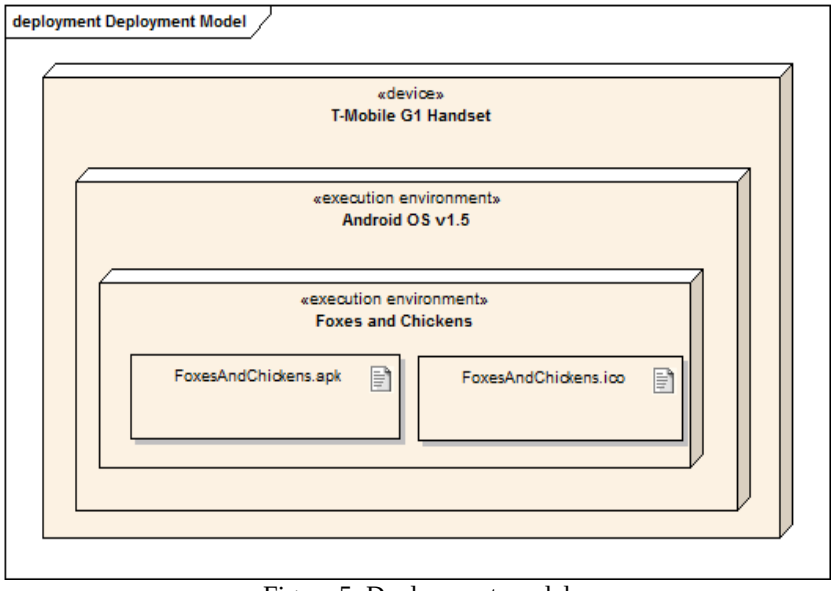

Figure 5. Deployment model

The final step, which is arguably one of the most important, is testing. It is of crucial importance to ascertain whether a software system does what it is supposed to do and fulfills the requirements set forth by the stakeholders of the system. Testing occurred throughout the development process and the findings helped shape development and game features. A few examples of the bugs found are listed below:

- Foxes, after completing a multi-jump, would land in invalid locations.
- Resetting the game would change the user selections from the previous launch.
- Several obscure issues that ended up being related to thread messaging.

At the time of this writing, there have been about 10,000~50,000 downloads of the *Foxes and Chickens* game from Google play (https://play.google.com/). The game is free to use and the source code will be made available to the public to help others learn about Android development. After we fixed issues found by users in the initial release of the game, users posted very positive comments on the game, such as 1) Very challenging, which is what makes it fun; 2) Challenging works flawlessly on Samsung Galaxy S; 3) Very smart game with cool graphics; 4) Fun game looks good; 5) Classic game - great when waiting for a flight or similar; works well.

V. CONCLUSION

In this paper, we presented an Android game named Foxes and Chickens that we have successfully designed and implemented for the Android mobile devices. Solid software engineering principles were utilized in the design and implementation of the Foxes and Chickens game. In addition, we performed extensive testing to validate different functionalities available in the game. Our testing and users' positive comments indicated that the Foxes and Chickens game met user expectations and is a fun game to play on Android mobile devices.

REFERENCES

[1] Open Handset Alliance, http://www.openhandsetalliance.com/
[2] Android Developers Web Portal, http://developer.android.com/index.html
[3] J. Arlow and I. Neustadt, UML 2 and the unified process (Addison-Wesley, MA: Boston, 2005).
[4] C. Fox, Introduction to software engineering design (Addison-Wesley, MA: Boston, 2006).
[5] R. Pressman, Software engineering (McGraw-Hill, NY: New York, 2005).

APPENDICES I: the Foxes and Chickens game rules

- Both players must move each turn. If a player is unable to move any pieces, that player loses. In the case of foxes, if a fox ever becomes trapped (i.e. cannot move), that fox is removed from the game.
- Foxes can move one unit in any direction, but in order to kill a chicken, the next unit beyond the chicken must be vacant. (This is akin to jumping in checkers).
- Foxes can perform multiple kills per turn provided that they do not violate any other rule (multi-jumps).
- Chickens can only move one unit in a forward or side direction. They may not move backwards. (Forward is defined as up, toward the chicken-pen).
- Movement is constrained to the directional paths as displayed on the game board.
- The chickens win by filling the pen with nine chickens. The foxes win by reducing the number of chickens on the board to a number less than nine.

APPENDICES II: Code snippet: scoring a move

```
//At intermediate difficulty, this method generates a score for the position passed in. For the Hard //setting, this is expanded to also take into account if a move will yield a future attack.
private int scoreMove(int curPos, int futPos, boolean hard) {
   int retVal = INVALID_MOVE;
   //foxes get a bonus for staying in the pen at hard difficulty
   int additive = futPos < (curPos-5) ? 1 : 0;
   if(fac.gameState[futPos] == 'x' || fac.gameState[futPos] == 'f')
      retVal = INVALID_MOVE;//impossible move
   else if (fac.gameState[futPos] == 's') {
      if(checkIndexForFutureKills(futPos)) {
         //movement which leads to a future kill (in one move)
         retVal = KILL_NEXT_ROUND;
         //if set to hard, give the applicable bonus
         retVal = hard ? (retVal + additive) : retVal;
      }
      else if(hard) {
         for(int i = 0; i < 8; i++) {
            int tempIdx = fac.movementMatrix[futPos][i];
            if(checkIndexForFutureKills(tempIdx))
               //movement which leads to a future kill (in two moves) is worth 2
```

```
                retVal = KILL_IN_2_ROUNDS;
            }
            //even if there are not future kill, this is still a valid move
            if(retVal == INVALID_MOVE)
                retVal = VALID_MOVE;
        }
        else
            retVal = VALID_MOVE;
    }
    else {//fac.gameState[futPos] == 'c' ... we need to check for a kill, or multiple kills
        if(checkIndexForKills(curPos, futPos) == false) {
            retVal = INVALID_MOVE;//no kill = invalid move
        }
        else {
            //by hitting this block we have a kill
            retVal = KILL_THIS_ROUND;
            //calculate the landing index
            int diff =  futPos - curPos;
            int landingIdx = futPos + diff;
            //check it for future kills
            if(checkIndexForFutureKills(landingIdx)) {
                retVal += KILL_NEXT_ROUND;//multiple kill
                if(hard) {
                    for(int i = 0; i < 8; i++) {
                        int tempIdx = fac.movementMatrix[landingIdx][i];
                        if(checkIndexForFutureKills(tempIdx))
                            retVal += KILL_IN_2_ROUNDS;
                    }
                }
            }
            retVal = hard ? (retVal + additive) : retVal;//if set to hard, give the applicable bonus
        }
    }
    return retVal;
}
```