**CBL Game Development**

*Assignment Pair 198*

Kolios, George (2113619)

Shenmei, Wu (2088681)

Table of Contents

[Game Overview: 3](#_Toc180964981)

[Topic Decision: 4](#_Toc180964982)

[Game Design 4](#_Toc180964983)

[Version Control (Git) 5](#_Toc180964984)

[References: 7](#_Toc180964985)

# Game Overview:

The game that was developed is called “Color Bingo”. We were inspired to make this game by two other popular games. The first one being “Connect Four”, which is a western game were two players compete with each other and take turns dropping a colored token into a 6x7 vertically suspended grid. The player who manages to drop four consecutive tokens of the same color orthogonally or diagonally wins. The second game was “Gobang”, also known as “Gomoku” which translates to “Five in a Row”, mostly popular in the east. In this game, two players compete by placing black and white stones on a 15x15 board and the first player who manages to line up five stones of the same color in an orthogonal or diagonal line wins.

Our game works similarly to Gobang, although we came up with the idea to make the game more interesting by making up some new rules and abilities and making them customizable. This means that the game’s main feature is making the game tailor-made to the users’ likings through personalization. When the game is initially ran, the users can click on the “Customize Rules” and they will be greeted with a new window pop-up with a set of options to choose from. More specifically, the users can; enable or disable abilities, that were developed by us; customize the grid size; and change their color.

The abilities that the users can choose from are:

1. Eliminate: Allows a player to eliminate one of the tokens placed by the opponent.
2. Obstacle: Creates a token which is neither of the players’ color. This means that a token will occupy a square and make it inaccessible to both players.
3. Swap: Allows a player to change one of the opponent’s tokens into their own.
4. Protect: Protects one of the player’s tokens from being eliminated or swapped by the opponent. (Indicated by changing the token’s border color to the opponent’s color.)
5. Spread: Places down a token and after 2 rounds, another token is placed randomly next to the original token. (If all squares are occupied, then no additional token is placed.)

Furthermore, the users can choose the size of the grid to be 5x5, 6x6, 7x7, 8x8, 9x9, or 10x10, and can also choose their tokens’ colors from a variety of 9 colors (red, orange, yellow, green, cyan, blue, purple, pink, brown).

Lastly, when a game is finished, its details will be written in a .txt file. By doing this, when a player clicks on the “Game History” button in the Main Menu, all games and their details will be visible linearly from the latest game to the first.

The game details include:

1. Game Count: A label which shows the number of the game.
2. Player Colors: A small colored box which depicts the players’ tokens’ colors.
3. Date Played: The date when the game was played.
4. Rounds (R): The total number of rounds that the game lasted.
5. Grid size (G): The number of rows or columns of the grid.
6. Rules: A label showing which rules were enabled in that game.
7. Winner: A label showing which player won, or which player gave up.

# Topic Decision:

## Game Design

The first topic that we chose for our game was Game Design. The reason we chose this topic was because we are both interested in Game Development as a career choice. The way we implemented Game Design in our game was mostly through the Rule Customization feature, although we found some other ways in which it was implemented.

Initially, the rules we created are classified as “Game Mechanics”, which is a large subtopic of Game Design. These rules make up the gameplay and create new ways in which the players can achieve victory. We created a total of 5 rules which is considerably an adequate amount of rules, as stated by multiple Research Papers in Game Design by the phrase “Less is more”.

One major aspect of the rules that we had to figure out was how the rules would interact with each other. For example, how would the Spread ability interact if the opposing player used Swap? In order to answer questions like these, we had to distinguish different cases for these questions to make the game fair for both players. For most cases, we wanted to encourage players to use all of the abilities without hesitation, so for the previous example, Spread would still create the initial player’s token although the original token was swapped by the opponent. Therefore, in general, rules would interact as expected with each other as shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Swap** | **Eliminate** | **Obstacle** | **Protect** | **Spread** |
| **Swap** | Not Possible | Eliminates the token if it’s the opponent’s color | Not possible | Not Possible | Not Possible |
| **Eliminate** | Not Possible | Not Possible | Places an Obstacle | Not Possible | Places a token that will cause Spread |
| **Obstacle** | Not Possible | Not Possible | Not Possible | Not Possible | Not Possible |
| **Protect** | Not Possible | Not Possible | Not Possible | Not Possible | Not Possible |
| Spread | Swap initial token, but Spread will still happen with the initial player token | Eliminate initial token, but Spread will still happen with the initial player token | Not Possible | Not Possible | Will cause Spread after the first player if and only if player 2 used Spread on a square that is not occupied |

*In this table, the abilities in the first row are used on the abilities in the first column.*

Another aspect of these rules is to increase player satisfaction by making them able to make the game tailor-made to their liking. Rule customization can enhance a user’s experience as they can chose how their game is going to be played. In turn, the game should also respond to the players’ actions corresponding to the chosen rules.

Additionally, player actions should have specific consequences. For example, players should think carefully before using an ability because abilities have a two round cooldown. That way, the players will have to consider whether or not they can use an ability (and which one) and strategize to make the best possible action. After all, “ColorBingo” is supposed to be a strategy game.

Moreover, we also decided to add a level of “Randomness and Nondeterministic Behavior” through the Spread function. As stated earlier, Spread randomly places a token of the same color in the box surrounding the initial token. That way, players have to consider whether they should risk using their ability to place two tokens at the cost of a turn, or else their action may lead to a blunder.

Lastly, another aspect which was implemented after the Rule Customization was the idea of program feedback. This means that the game is supposed to give an explanation or show something to a player after an action. The most obvious case of feedback is during the game, when a player makes a move, a label will state that player’s action. (i.e. Player 1 used Swap on (3, 4).) The example states the exact action, by the specified player, on which coordinates in the grid.

## Version Control (Git)

The second topic we choose is doing Version Control in Git. For our game, we decided to use GitHub to work collaboratively and to maintain a history of all changes made during the development phase.

To do version control, Git applies Distributed version control systems (aka DVCS). DVCS use a [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer) approach to version control, as opposed to the [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server_model) approach of centralized systems. Distributed revision control synchronizes repositories by transferring [patches](https://en.wikipedia.org/wiki/Patch_(Unix)) from peer to peer. There is no single central version of the codebase; instead, each user has a working copy and the full change history. Advantages for applying DVCS system in Git are as follows:

* Allows users to work productively when not connected to a network.
* Common operations (such as commits, viewing history, and reverting changes) are faster for DVCS, because there is no need to communicate with a central server. With DVCS, communication is necessary only when sharing changes among other peers.
* Allows private work, so users can use their changes even for early drafts they do not want to publish.
* Working copies effectively function as remote backups, which avoids relying on one physical machine as a single point of failure.
* Allows various development models to be used, such as using [development branches](https://en.wikipedia.org/wiki/Branching_(version_control)#Development_branch) or a Commander/Lieutenant model.

Branching is another crucial feature of Git. Different branches in Git may represent different functions or different parts members are in charge of in the system. Once a branch is finished and checked, it will be merged into the main branch later. This kind of isolation property of branching enables members to have enough freedom to work on their own part, also it protects the whole system from potential threats from mistakes or attacks. By working with branches in Git for CBL project, we learned how to separate our goal into small parts and how to manage our expectation along with progress timeline.

Merge in Git is powerful function we used based on branching system. Merge allows us to add a branch into another. What usually happened is both members need to work on the same version at the same time instead of one by one. To deal with common commitment origin problem, Git could examine branches first and give us a merging solution automatically. Auto merging doesn’t work when Git finds out different adjustment for the same part in both commitments, so we need to merge these changes manually. Moreover, merging enables us to track back to any point where possible mistakes or accidents occurred since merging history is traceable.

Finally, Git gives different access to different groups of members respectively. Since the access to different parts of the code is limited, it is also convenient to do version control in Git. In our project, full access to the code was given to both of us in order to work collaboratively and to share the workload equally and effectively. The full Git directory can be accessed through [here](https://github.com/SammiDiamond/CBL-project).

# References:

Almeida, M.S.O. & da Silva, F.S.C. (2013) 'A systematic review of game design methods and tools', in Anacleto, J.C. et al. (eds) *Entertainment Computing – ICEC 2013*, Springer, Berlin, pp. 3-10. doi: 10.1007/978-3-642-41106-9\_3.

Dormans, J. (2012) *Engineering emergence: applied theory for game design*. Teste de Doutorado. Amsterdam University of Applied Sciences.

Zolkifli, N.N. & Ngah, A. (2018) 'Version control system: a review', *School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, 21030 Kuala Nerus, Terengganu, Malaysia*.