# Outline for developing skyjo in C++

Martin Gjorgjevski

November 24, 2024

So far we have implemented the deck in a class `Deck` which has two private vector members: `deck_down`, which represents the pile of cards from which the players will draw, and `deck_pile`, which represents the cards that have been discarded from the players. We also have a helper class titled `Hand` which is supposed to represent the cards of the player. It's members are $3 \times 4$ `int array` titled `hand` which holds the values of the cards, and $3 \times 4$ `bool array` titled `mask` which tells us whether a particular card is face up or face down. In addition, we have a class `Player` , which contains as private members a `Hand` instance and an identifier `id` .

At runtime, we ask for the number of players, and dynamically allocate memory for the number of players entered. These are pointers to `Player` of the input length. The deck is then instantiated, and cards are distributed to all players. At the initial stage of the game we need to ask the players to reveal two cards, after which the game can commence. At this stage of development, we have implemented these elements. The next objective will be to run the game.

## 1 Objectives

Once the cards are distributed, and the player that goes first is determined, we need to loop around the players, asking them to make a move. So, there are two things to figure out.

**How to loop around the players in a round-robin fashion ?** In the code, each player, represented by `Player` class, gets a dynamically allocated memory address at runtime. One way to track which players turn it is, is to start a clock which will count modulo the number of players, and the time of the clock will tell whose turn it is. Let's implement and test this before we implement more game mechanics. Ok, after some off by 1 errors, it seems that we've made our idea work. Now, after initialization, the game loops through the players without ending. Lets implement game mechanics

**What can a particular player do on his turn?** The player starts his turn by drawing a card. He can either draw face down card from `deck_down` or the last discarded card from `deck_pile` . Upon drawing a card, the game continues as follows. If he drew a card from the pile `deck_pile` , he chooses a card from his hand (could be a face-up or face-down card) and discards it, and in it's place he puts down the drawn card. Then his turn ends.

If he drew a card from the face down pile `deck_down` , then he may act as in the previous case, i.e. discards a card from his hand and in its slot he puts down the drawn card and end his turn. Alternatively, he may discard the drawn card. In this case, he has to choose a card that is face down, and reveal it and then his turn ends.

The first player that has all of the cards in his hand face up signifies the last cycle of the game. There remains one single turn for everyone, after which they reveal their hands and the player with

the lowest score wins. There are more nuances to the rules, but for the time being let us focus on implementing these mechanics.

Another thing, that will soon come up is what to do when the face down pile `deck_down` is exhausted. We will come back to this.

**Exhausted deck** Ok, we've made some progress. The game now runs, as intended (or at least no bugs are evident to me at this point). However, there are two problems that need to be addressed now. Firstly, we need to address the possibility of exhausting the face down pile `deck_down` . If we try to draw from an empty vector, bad things may happen. In real life, when it comes to this scenario, we keep the last card as usual, but the remainder of the face up pile (`deck_pile` in our implementation) are reshuffled and are used again as a face down pile i.e. `deck_pile` is shuffled and *moved* to `deck_down` . Next, how the game terminate? We will discuss this after solving the exhausted deck problem.

**The column rule** Figuring out how to move properly handle the case when there are no more cards in `deck_pile` took some time, but it works correctly after some tests. Now, the game does not have win condition implemented, meaning that even when all the cards are revealed, the game can go on and on.... Before we implement win condition verification, we need to implement the rule of columns: if three of a kind is formed in a column, the column turns to zeros (this applies for negative cards $-1$ and $-2$ as well). The cards are *burned*, they all go on the `deck_pile` , with the common column value ending up on top. Moreover, the player is no longer allowed to stack cards in a column that he burned (not that a person would do that, but still) Before we implement the end condition, we also need to implement this feature into the game. The column rule was fairly simple to be implemented. After each turn you check whether the cards in the column are face up and if they show the same value. If this is the case then you set that column to 0 and push the cards onto `deck_pile` .

**Win conditions** Last thing to do to have a working implementation is to check whether a certain player has opened his hand entirely. Then you need a way to break out of the while loop. This could pose some off by one problems as the deck reshuffling implementation did, but it shouldn't be too bad. At this point, the game essentially is finished, however, some input validation is needed, and more features can also be added. We will continue next time, it's enough for today `:-)`

## 2 Game Mechanics

When a player draws a cart he may want to keep it or discard it. To keep track of the discarded deck, we first need to process the decision of the player. We need to keep track of whose player's turn is it. At some point, before the game begins we ask how many players there are. We instantiate each player and keep looping around until a critical condition is met, i.e. when some player has all cards revealed, at which point one last turn remains for the other players, after which the scores are tallied up and the winner is declared. The column rule - if you collect three cards of identical values in a column, you must "discard" that column, i.e. for practical purposes each entry in the column is replaced by 0. This applies to negative values as well, so a column of three minus one cards will again reduce to a column of zeros.

## 3 Additional Concerns

Supposing that the face down deck is emptied out, the game continues by shuffling the discarded deck (face up), placing it face down, and reusing it as the face down deck. This will eventually need to be implemented.

All players should have access to the information that are available in the board game, i.e. they can see the hands of other players, as well as the discarded deck (For example, you might decide to gather a column of 12's later in the game; the knowledge of how many 12's have passed around will influence this decision - interestingly, this is slightly harder to do in the actual board game, as going through the discarded deck is frowned upon by the other players).

## 4 Game termination

After a player turn ends, we need to check if the end condition is met, i.e. whether his entire hand is face up. If this is the case, the game terminates the next time the player pointer is pointed to him, and the winner is declared.