APS106 Project - A Simple Version of CHECK OUT LINE

**Introduction**

Team Jimmu, consisting of Anthony Gao(), Richard ShangGuan, Jimmy Tieu and ZipengCai(999780367), has been assigned to create a simple version of the game “Check Out Line” as their APS106 project. In this game, a player (human or computer) must try to score as many points as possible by eliminating same-coloured shopping carts, labeled as letters corresponding to the colour, on a randomly generated or preset grid. The player tries to attain the highest score possible, and the game ends when no more moves are available to be made. The most recent game’s statistics will be stored in a log. Interesting features of the code are listed in Appendix A. The code was developed and tested using Codeblocks and its included GCC compiler.

**How to play**

To play the game, the player may use Codeblocks to open and run the game's code. When run, the game will prompt player to enter their name, then choose between a computer generated or custom grid. If the player chooses a computer generated grid, the player will be prompted to enter 2 integers, from 6 to 36, corresponding to the x- and y- axis, to determine the size of the grid. Should the player use a custom grid, the player must enter the path to a .txt file containing the grid. The player then chooses either to play the game or let an A.I play.

The objective of the game is to score as much points as possible. Points are scored by clearing an area of shopping carts of the same colour. This is done by selecting a coordinate on the grid. The game will then clear all shopping carts of the same colour that are adjacent to it and the other shopping carts around it until an area of same-coloured shopping carts has been cleared. The game will only accept coordinates that contains shopping carts of the same colour directly adjacent to it. Points are then awarded based on the square of the number of shopping carts cleared that turn. Each turn, points are scored and added until there are no valid moves left to be made. The game keeps track of every coordinate inputted and the points accumulated at that point in a log file within the game folder. Once the game determines that no more moves can be made, the game ends and the player's total score will be shown.

**Coding**

The code begins with listing the headers and global variables to be used in the program. Functions are then written, their name and what they do listed below:

* return\_int: Changes user inputted coordinates into program accepted ones
* mate\_grid: Copies elements of the alternate grid to main grid
* caps: Capitalizes characters
* expand: Used to highlight and count elements of grid to be removed(Appendix B)
* possible: Checks of a move is valid
* collapse: Removes elements of grid and shifts elements when necessary
* timestamp: Stamps the beginning of each game started in the log
* generate\_grid: Used to generate the grid
* read\_grid: Used to read grid from a file to use
* print\_grid: Prints grid into terminal
* play: Used for human play
* ai\_play: Used for A.I play
* start: Initializes game, lets player choose between human or A.I play
* main: Opens log files, asks for player name

The program starts off in 'main', which opens the log file and prompts player to enter their name. It then calls 'start', which asks user to choose between a computer generated or custom grid, and human or A.I play. It then calls 'generate\_grid' and either 'play' or 'ai\_play', depending on what was chosen earlier. The grid is pseudo-randomly generated, with higher chances of the previously generated element being generated next, allowing for more options.

When 'play' is called, the program prompts the user every turn to enter a set of coordinates. Those coordinates are sent to 'expand' and 'collapse', which mark elements in an alternate grid used for testing and remove marked elements, respectively. 'collapse' also shifts elements down and over when there is an empty space below or column to the left, and updates the main grid after elements have been eliminated by calling 'mate\_grid'. Every turn, the coordinates entered are inputted into a log file and points obtained this turn are added to the total score. This process is repeated until no more moves are available, which is done calling the function 'possible' after every turn. The final score is shown and the player is sent back to 'start'.

When ‘ai\_play’ is called, the program plays the game using the written algorithm. It works by first making a hidden grid, used to determine which coordinate to choose. ‘expand’ is called to the hidden grid on every valid coordinate to check which coordinate will give the highest points for that turn. That coordinate will be used to obtain points that turn. The program then proceeds as detailed above.

**Conclusion**

This project has been a valuable learning experience for its team members. Jimmy Tieu, as team leader, wrote the report and ensured the team's work quality was up to par and done on time. Anthony Gao, as lead programmer, wrote the key functions that made the program work as intended. Richard Shang Guan, as A.I developer, developed the algorithm for the program's computer play mode. Zipeng Cai, as head of communications and quality assurance, ensured that the program had no errors and followed all instructions given by instructors.

**Appendix A - Neat features**

There are many minor details of the code, all of which contribute to the game's quality, listed below.

* A special character, "!", that when inputted into x and y by the user during a turn, will terminate the program
* Ability to output the shopping carts as colours in the terminal
* Many restrictions to prevent user from crashing program
* Asks user for name, as a "quality of life" addition
* Grid generation made for a slightly higher chance of same-coloured shopping carts to be together

**Appendix B - The 'expand' function**

The recursive function 'expand' is the most important part of the program, as it is used every turn to mark which functions to remove. It first checks if the coordinates inputted exist on the grid. If it doesn't, it exits the function. It then capitalizes the element in the given coordinate to mark it. If the function was called from 'possible', then it immediately returns to it at this point for efficiency. If not, the function then checks the elements directly vertical and horizontal to it, and if they are the same, calls 'expand' on that coordinate. This continues until the function finishes checking that there are no more elements to capitalize. It returns 'steps', which was incremented every time 'expand' ran, which indicates the number of elements removed that turn and thus used to calculate points obtained that turn.