Assembly Language
Final Project

Group 30

111502509 鄭秉軒

111502510 林文仁

111502511 林暐勛

111502512 郭拓海

TOPIC

雙人打方塊 Two-player Arkanoid

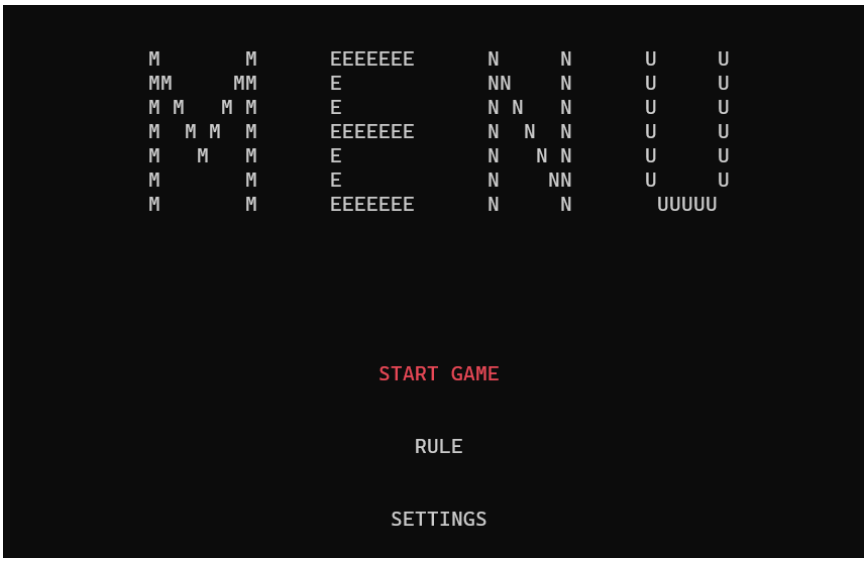
Steps of program execution flow

There are three asm program to complete our final project.

1. main.asm: It shows the menu of the game, including rules and setting.
2. final.asm: It represents the first game mode, “Battle”.
3. final1.asm: It represents the second game mode, “Teamwork”.

---In main.asm---

> There is a word “MENU” which built and combined with many characters, using loop and invoking procedure “WriteConsoleOutputCharacter”.

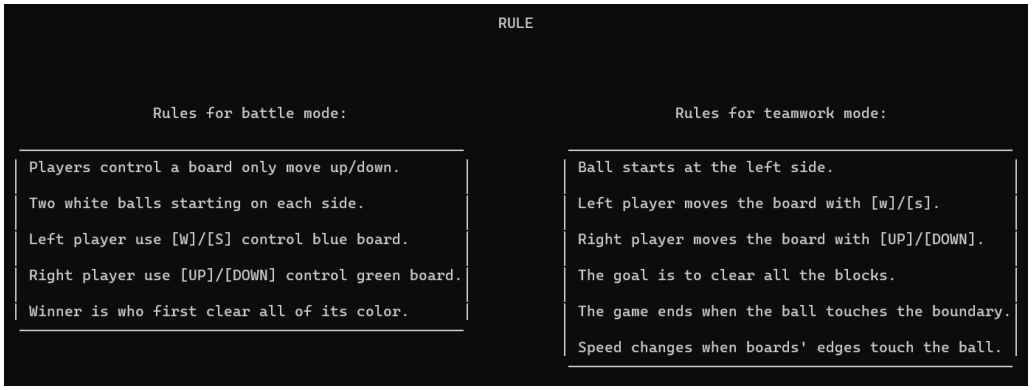


> Using WSAD key on the keyboard to decide which function be used. As the state 0 represent the starting of game, the game mode which players choose would be started. State 1 represent the rule interface, telling players about rules of two different game mode. State 2 represent setting interface, where players can choose key A and D to change the game mode. (Game mode would be set as “BATTLE” at the initial stage.)

> Pressing ENTER key to confirm your choice, which appears with red word.

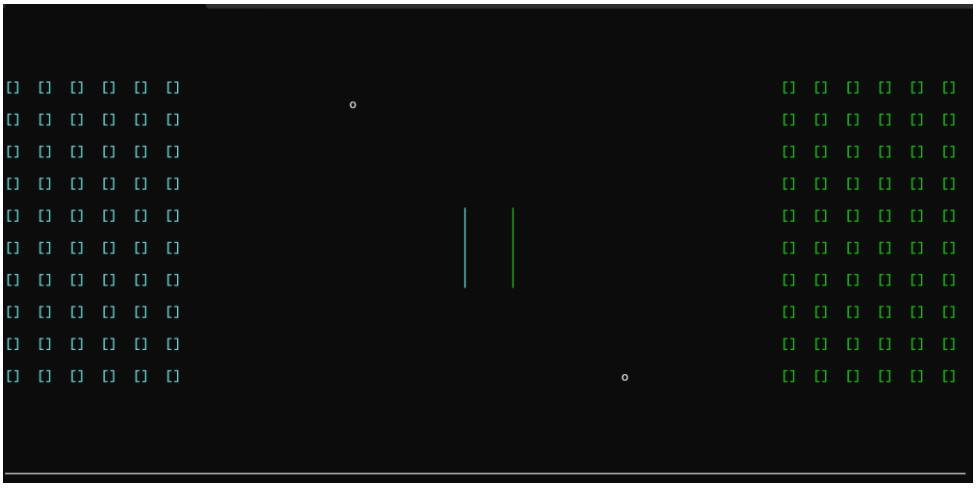
> Also pressing ENTER key to turn back from both rule and setting interfaces. Our program will remember the choice players made.

*Rule interfaces: Rules of BATTLE is on the left, of TEAMWORK on the right.

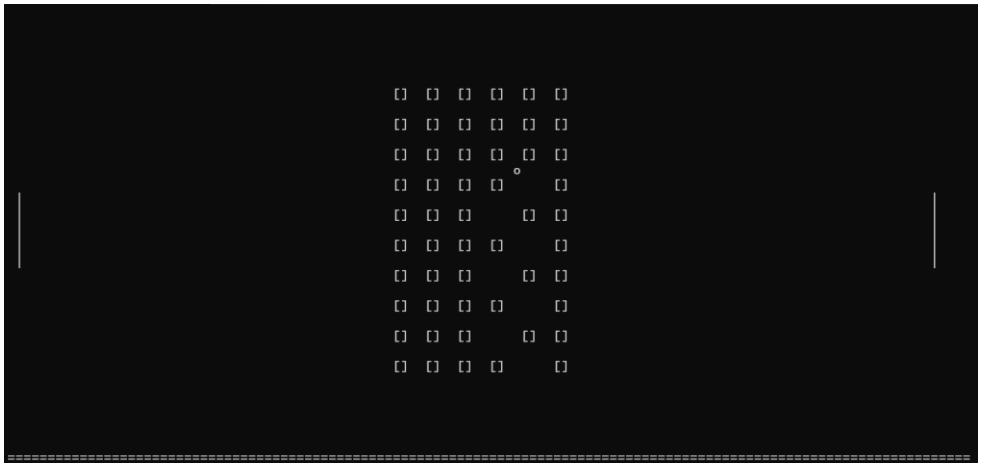


*Setting interfaces: The place for players to choose the game mode.

*A screenshot of BATTLE mode. (Two players with different colors)



*A screenshot of TEAMWORK mode. (Two players on left and right side)



---In final.asm---

*There are many different procedures combined to implement the game.
*Followings are the steps from the beginning to the end in one loop. When starting the game, programs would keep looping a series step of what should be done in a microsecond of the game.

~~~~~~

> Draw two boards which are controlled by two players.
> Draw two balls at the location of this moment on the screen.
> Draw the moving boundary of two balls and two boards.
> Draw the leftover blocks on two different players sides.
> Judging two balls' movement trajectory, check if their collision with blocks or decide the location where balls would appear at the next second.
> Delay the program by looping empty program for millions of times.
> Wait for players input the key and control the board. If there's no input keys, then return null and keep going on the next looping run.
> Check if the condition is conformed to the ending of the game. That is, one of the players' blocks are all destroyed, consider as the winner.

---In final1.asm---

*There are many different procedures combined to implement the game.
*Followings are the steps from the beginning to the end in one loop. When starting the game, programs would keep looping a series step of what should be done in a microsecond of the game.

~~~~~~

> Draw two boards which are controlled by two players.
> Draw the ball at the location of this moment on the screen.
> Draw the moving boundary of the ball and two boards.
> Draw the leftover blocks at the middle of the screen.
> Judging ball's movement trajectory, check if its collision with blocks or decide the location where the ball would appear at the next second.
> Delay the program by looping empty program for millions of times.
> Wait for players input the key and control the board. If there's no input keys, then return null and keep going on the next looping run.
> Check if the condition is conformed to the ending of the game. That is, all of the blocks at the middle of the screen are all destroyed, consider as mission completed. Or the ball moves out of the screen, such as boundary, consider as mission failed which would shows the score of the game. (Calculate fractions: Break one block gets one points)

Implement our functions

#Invoking the procedure “GetStdHandle” to get the output console handle.
#Invoking the procedure “SetConsoleCursorInfo” to hide the cursor.
#Invoking the procedure “Clrscr” to clear the screen from the beginning of the series step of one loop term.
#Invoking the procedure “WriteConsoleOutputAttribute” to change the color of the coordinate location and “WriteConsoleOutputCharacter” to print out the characters which represent balls, boards as well as boundaries.
#Also, with the procedure “WriteConsoleOutputCharacter” to print out the sentence of words that are already written by us, such as rules or the epilog.
#Call the procedure “Readkey” to check if there is any input key from the players. If yes, then the zero flag might not be set up, so it jumped to other label to implement programs which should be done when press corresponding input keys. Otherwise, it keeps going onto the next step. For example, when players pressing W key, it would force program to move up the board.
#There is a delay procedure in the program to delay the refresh of the screen, and slower the gaming time at the same time.
#Setting balls, blocks and boards as their corresponding coordinate is the way how the program controls the movement of balls and boards, or the existence of leaving blocks. Adding or subtracting the coordinate by the fixed value could made the movement stabilize and physical. Changing the value into blank represent the collision with the block.
#Assume two variables to monitor if it is satisfied to the ending condition at any time. Then jump to label that shows the finishing interfaces.
#“EXTERN “name”@0:PROC” and “call “name”@0” is used to call the different asm program. The advantage is that programs would be more logical and easier to tidy and revise.
#From the beginning of the main program of final.asm and final1.asm, there are several lines of program in order to initialize various of data to the original statement. Making the game restartable.
#Every starting of the program contains the action of pushing registers into stacks, and every ending of the program contains the action of pop values of register out of stack. The purpose of those action is to make sure the value would not change weirdly during some procedure.

NOTES

• Our final project had require environment of chcp 437, which had to change the chcp value from CMD, or there would be lots of weird symbol on the screen.

• The whole project or said to be game, controls completely by keyboard. The player on the left used WASD keys to control the board. And another player on the right used arrow keys to control the board. There's no need of using cursor for both two players.

• Two different game mode are both able to replay. And the default game mode is "battle".

• When at the menu interfaces, if you are sure to choose the choice which the words are with red color. Then press 'ENTER' key to keep going on for the next step, such as turned back to the menu interfaces or went into the corresponding interfaces.

Variable Explanation

---In main.asm---

- state BYTE 0 ; Control which function players had chosen.
- boolvalue BYTE 0 ; Preserve 1 if the ENTER key is pressed.
- gm BYTE 0 ; Save the game mode chosen by players.
- InfoCursor CONSOLE_CURSOR_INFO <1,0>
  ; Assign the size and the visibility of the cursor.
- consoleHandle DWORD ? ; Store the output handle.
- cellsWritten DWORD ? ; Receive useless number by PROCs.
- bytesWritten DWORD 0 ; Supply nonchanged number to PROCs.
- state0 BYTE “START GAME” ; Show several words.
- state1 BYTE “RULE” ; Show the words.
- state2 BYTE “SETTING” ; Show the words.
- gmode0 BYTE “GAMEMODE” ; Show the words.
- gmode00 BYTE “-USING [A][D] OR [LEFT][RIGHT]”; Show serval words.
- gmode1 BYTE “BATTLE” ; Show the words.
- gmode2 BYTE “TEAMWORK” ; Show the words.
- xy0 COORD <55,16> ; Print out state0 from this position.
- xy1 COORD <58,19> ; Print out state1 from this position.
- xy2 COORD <56,22> ; Print out state2 from this position.
- xy3 COORD <56,5> ; Print out gmode0 from this position.
- xy4 COORD <45,7> ; Print out gmode00 from this position.
- xy5 COORD <37,13> ; Print out gmode1 from this position.
- xy6 COORD <76,13> ; Print out gmode2 from this position.
- xy7 COORD <58,5> ; Print out state1 from this position.
- attri WORD 10 DUP(0Ch) ; Set the character color as red.
- line1 BYTE 51 DUP(0C4h) ; Draw a line as the character “_”.
- line2 BYTE 0B3h ; Draw a line as the character “|”.
- Lxy1 COORD <4,12> ; Print out line1 from this position.
- Lxy11 COORD <4,22> ; Print out line1 from this position.
- Lxy2 COORD <3,13> ; Print out line2 from this position.
- Lxy22 COORD <54,13> ; Print out line2 from this position.
- Lxy3 COORD <66,12> ; Print out line1 from this position.
- Lxy33 COORD <66,24> ; Print out line1 from this position.
- Lxy4 COORD <65,13> ; Print out line2 from this position.
- Lxy44 COORD <116,13> ; Print out line2 from this position.
- rule1 BYTE “Rules for battle mode:”

- rule2 BYTE “Players control a board only move up/down.”
- rule3 BYTE “Two white balls starting on each side.”
- rule4BYTE “Left player use [W]/[S] control blue board.”
- rule5 BYTE “Right player use [UP]/[DOWN] control green board.”
- rule6 BYTE “Winner is who first clear all of its color.”
- rule7 BYTE “Rules for teamwork mode:”
- rule8 BYTE “Ball starts at the left side.”
- rule9 BYTE “Left player moves the board with [w]/[s].”
- rule10 BYTE “Right player moves the board with [UP]/[DOWN].”
- rule11 BYTE “The goal is to clear all the blocks.”
- rule12 BYTE “The game ends when the ball touches the boundary.”
- rule13 BYTE “Speed changes when boards' edges touch the ball.”

  ; Rule1 to rule13 will show on the screen on the rule interfaces.
- rulexy1 COORD <19,10> ; Print out rule1 from this position.
- rulexy2 COORD <5,13> ; Print out rule2 from this position.
- rulexy3 COORD <5,15> ; Print out rule3 from this position.
- rulexy4 COORD <5,17> ; Print out rule4 from this position.
- rulexy5 COORD <5,19> ; Print out rule5 from this position.
- rulexy6 COORD <5,21> ; Print out rule6 from this position.
- rulexy7 COORD <78,10> ; Print out rule7 from this position.
- rulexy8 COORD <67,13> ; Print out rule8 from this position.
- rulexy9 COORD <67,15> ; Print out rule9 from this position.
- rulexy10 COORD <67,17> ; Print out rule10 from this position.
- rulexy11 COORD <67,19> ; Print out rule11 from this position.
- rulexy12 COORD <67,21> ; Print out rule12 from this position.
- rulexy13 COORD <67,23> ; Print out rule13 from this position.
- M BYTE “M” ; Draw a character “M”.
- E BYTE “E” ; Draw a character “E”.
- N BYTE “N” ; Draw a character “N”.
- U BYTE “U” ; Draw a character “U”.
- resultPos1 COORD <36,3> ; Print out M from this position.
- resultPos2 COORD <36,3> ; Print out M from this position.
- resultPos3 COORD <40,7> ; Print out M from this position.
- resultPos4 COORD <44,3> ; Print out M from this position.
- resultPos5 COORD <51,3> ; Print out E from this position.
- resultPos6 COORD <52,3> ; Print out E from this position.
- resultPos7 COORD <52,6> ; Print out E from this position.
- resultPos8 COORD <52,9> ; Print out E from this position.

- resultPos9 COORD <64,3> ; Print out N from this position.
- resultPos10 COORD <64,3> ; Print out N from this position.
- resultPos11 COORD <70,3> ; Print out N from this position.
- resultPos12 COORD <77,3> ; Print out U from this position.
- resultPos13 COORD <77,9> ; Print out U from this position.
- resultPos15 COORD <83,3> ; Print out U from this position.

~~~~~~

---In final.asm---

- len=1 ; The length that used to draw two boards.
- winlen=6 ; The length that used to print words “WINNER”.
- loselen=5 ; The length that used to print words “LOSER”.
- blocklen=24 ; The length that used to draw blocks.
- blsize=6 ; Limit the width of how many blocks.
- boundwid=120 ; The length that used to draw the boundary.
- consoleHandle DWORD ? ; Store the output handle.
- xyBound COORD <120,27> ; Limit the boundary of two boards.
- xyBBound COORD <116,28> ; Limit the boundary of two balls.
- xyPos1 COORD <57,12> ; Assign board1 to this position.
- xyPos2 COORD <63,12> ; Assign board2 to this position.
- blPos1 COORD <0,4> ; Print out blocks1 from this position.
- blPos2 COORD <97,4> ; Print out blocks2 from this position.
- endxy1 COORD <30,14> ; Print out “WINNER” from this position.
- endxy2 COORD <90,14> ; Print out “LOSER” from this position.
- endxy3 COORD <60,0> ; Print out a line “|” from this position.
- boundary1 COORD <120,0> ; Print out bxy1 from this position.
- boundary2 COORD <0,28> ; Print out bxy2 from this position.
- rPos COORD <47,28> ; Print out ending from this position.
- bxy1 BYTE 0B3h ; Draw the boundary as character “|”.
- bxy2 BYTE 120 DUP(0C4h) ; Draw the boundary as character “_”.
- cellsWritten DWORD ? ; Receive useless number by PROCs.
- bytesWritten DWORD 0 ; Supply nonchanged number to PROCs.
- CursorInfo CONSOLE_CURSOR_INFO <1,0>
  ; Assign the size and the visibility of the cursor.
- board BYTE 0B3h ; Draw both two boards as character “|”.
- ballxy1 COORD <39,15> ; Assign ball1 to this position.
- ballxy2 COORD <81,14> ; Assign ball2 to this position.
- speed1 COORD <2,1> ; Set the original speed of the ball1.
- speed2 COORD <-2,-1> ; Set the original speed of the ball2.
~~~~~~

- one BYTE 0h ; Detect if player-one's blocks all destroyed.
- two BYTE 0h ; Detect if player-two's blocks all destroyed.
- ball1 BYTE 6Fh ; Draw a ball as the character "o".
- ball2 BYTE 6Fh ; Draw a ball as the character "o".
- block11 BYTE blsize DUP(5Bh,5Dh,20h,20h)
  ; Draw blocks as the character "[] ", and it contains a line of six blocks.
- …block110/210 BYTE blsize DUP(5Bh,5Dh,20h,20h) ; Same as above.
- attributes0 WORD blocklen DUP(0Bh) ; Set the color of blocks as blue.
- attributes1 WORD 0Ah ; Set the color of board as green.
- attributes2 WORD (blocklen-2) DUP(0Ah) ; Set blocks' color as green.
- attributes3 WORD 0Bh ; Set the color of board as blue.
- attributes4 WORD 6 DUP(0Ah) ; Set the character color as blue.
- attributes5 WORD 6 DUP(0Bh) ; Set the character color as blue.
- winner BYTE "WINNER" ; Show the words.
- loser BYTE "LOSER" ; Show the words.
- ending BYTE "Press 'ENTER' Back to Menu" ; Show sentence of words.

~~~~~~

---In final1.asm---

- len=1 ; The length that used to draw two boards.
- blocklen=24 ; The length that used to draw blocks.
- blsize=6 ; Limit the width of how many blocks.
- scMes BYTE "YOUR SCORE" ; Show the sentence of words.
- winMes BYTE "MISSION COMPLETE" ; Show the sentence of words.
- loseMes BYTE "MISSION FAILED" ; Show the sentence of words.
- backMes BYTE "Press 'ENTER' Back to Menu" ; Show the words.
- scAttr WORD 13 DUP(0Eh) ; Set the color of scMes as yellow.
- loseAttr WORD 16 DUP(0Ch) ; Set the color of loseMes as red.
- winAttr WORD 18 DUP(0Ah) ; Set the color of winMes as green.
- resultPos COORD <50,15> ; Assign the message screen position.
- score BYTE 0 ; Counting how many blocks are hit.
- consoleHandle DWORD ? ; Store the output handle.
- xyBound1 COORD <116,28> ; Limit the boundary of the game.
- xyPos1A COORD <1,12> ; Assign board1 to this position.
- xyPos2B COORD <115,12> ; Assign board2 to this position.
- blPos1C COORD <48,5> ; Print blocks from this position.
- cellsWritten DWORD ? ; Receive useless number by PROCs.
- CursorInfoC CONSOLE_CURSOR_INFO <1,0>
  ; Assign the size and the visibility of the cursor.
~~~~~~

- boardC BYTE 0B3h ; Draw the boards as the character “|”.
- ballxy1C COORD <31,16> ; Set the original position of the ball.
- speed1C COORD <2,1> ; Set the original speed of the ball.
- speed2C COORD <2,3> ; Speed when ball impact board edge.
- spp BYTE 0 ; State of the ball’s speed.
- ball3 BYTE 6Fh ; Draw the ball’s shape as character “o”.
- LoseFlag BYTE 0h ; Detect if the ball runs out of boundary.
- bou BYTE “===…==” ; Show the shape of bottom boundary.
- block31 BYTE blsize DUP(5Bh,5Dh,20h,20h)
  ; Draw blocks as the character “[] ”, and it contains a line of six blocks.
- …block310 BYTE blsize DUP(5Bh,5Dh,20h,20h) ; Same as above.

Code Descriptions

---In main.asm---

1. delayer PROC: Delay and slow down the whole program.
2. keyinput PROC: Read keys players input to control position of boards.
3. main PROC:
   First three lines are used to get console handle and invisible cursor.
   Line 125 calls the procedure “Clrscr” to clear the screen.
   Labels “drawM1”…“drawU3” are used to draw the words “MENU”.
   Line 244~252 control the choice that players might have chosen.
   Line 253~255 prints out the functions which players could choose.
   Line 256 calls the procedure “delayer” to slow down the program.
   Line 257~272 determine if players give the enter key to their choice.
   If yes, jumps to the corresponding label of different choice.
   Line 273 calls instruction in Irvine32 library and detect the input key.
   If there’s an input, jump to label “L1” and calls procedure “keyinput”
   Line 120~279 will run as loop until program jumps to “choice label”.
   Label “coop” calls out the final1.asm, and starts the teamwork game.
   Label “vs” calls out the final.asm, and start the battle game.
   Label “rule” prints out all the rules about two different game modes.
   Label “drawL” and “drawL1” draw the surrounding lines for the rules.
   It would wait for the enter key to return back to the menu interfaces.
   Label “setting” prints out the current settings of game modes.
   Players could change game mode by pressing corresponding keys.
   It would wait for the enter key to return back to the menu interfaces.

No need to initialize data or variables, because they wouldn’t change.

---In final.asm---

1. stopp PROC: Delay whole program and slow down the gaming time.
2. drawblocks PROC: Draw the blocks at assigned position row by row.
3. listener PROC: Read keys players input to control position of boards.
   If there's no input, keep going on the next step in the loop term.
4. ballrun1 PROC: Adjusting the movement of the ball1, and checking if the ball collision with blocks at the same time.
5. ballrun2 PROC: Adjusting the movement of the ball2, and checking if the ball collision with blocks at the same time.
6. final PROC:

   First three lines are used to get console handle and invisible cursor.
   The next four lines are used to initialize two boards.
   And the next four lines are used to initialize the position of blocks.
   And the next four lines are used to initialize two balls.
   And the next four lines are used to initialize the speed of balls.
   Line 727,728 are used to initialize the detectors.
   Labels "INITIAL1~20" are used to initialize all the blocks.
   Label "drawB" and "drawB2" draw two boards on each side.
   Line 997,998 draw two balls on the screen.
   Label "L8" draws the game boundary on the screen.
   Line 1012 calls the procedure "drawblocks" to draw all the blocks.
   Line 1013 calls the procedure "ballrun1" to adjust ball1's movement.
   Line 1014 calls the procedure "ballrun2" to adjust ball2's movement.
   Line 1015 calls the procedure "stopp" to slow down the program.
   Line 1016~1021 determine if the game ends, that is, one player wins.
   If it satisfied the condition of ending, jump to label "win1" or "win2".
   Line 1022 calls instruction in Irvine32 library and detect the input key.
   If there's an input, jump to label "L2" and calls procedure "listener".
   Line 974~1035 will run as loop until program jumps to "win1"/"win2".
   Label "win1" turns to ending interfaces and prints winner as palyer1.
   Label "win2" turns to ending interfaces and prints winner as palyer2.
   Label "L9""L10" wait for the 'ENTER' key to jump to label "END_FUNC".
   Label "END_FUNC" at the line 1089 will call back to the main.asm.

   ⇨ The game mode of "Battle".

Initialize data or variables for players to replay the game again.

---In final1.asm---

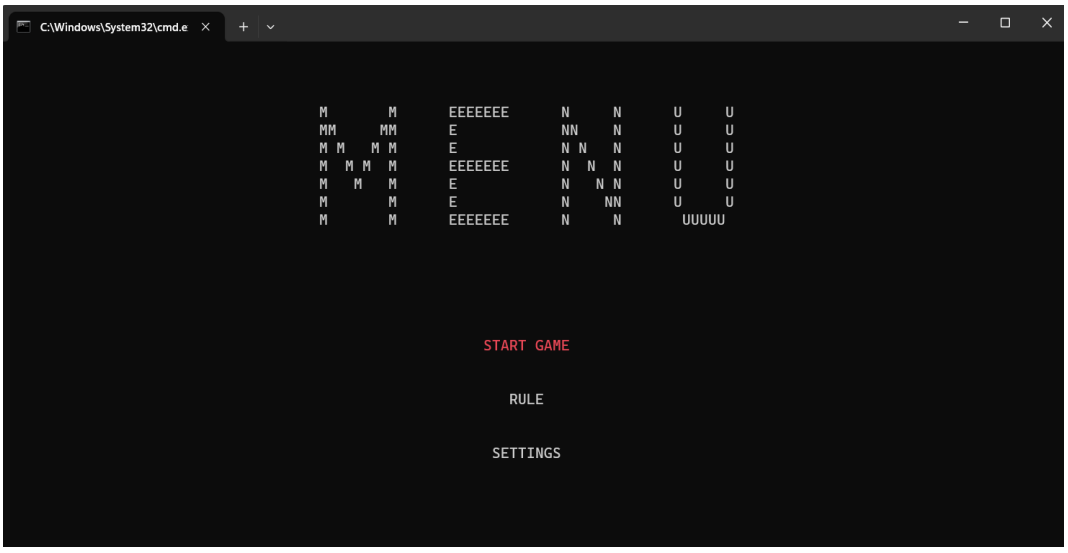
1. stoppp PROC: Delay whole program and slow down the gaming time.
2. drawblocksC PROC: Draw the blocks at assigned position row by row.
3. listenerC PROC: Read keys players input to control position of boards.
   If there's no input, keep going on the next step in the loop term.
4. ballrun PROC: Adjusting the movement of the ball, and checking if the ball collision with blocks at the same time.
5. LOSE PROC: Print the lose message and the score on the screen.
6. WIN PROC: Print the win message and the score on the screen.
7. final1 PROC:
   First four lines of codes are used to initialize two boards.
   The next two lines are used to initialize the ball.
   Labels "INITIAL1~10" are used to initialize all the blocks.
   Line 501 hides the cursor on the screen.
   Line 508 draws the bottom boundary.
   Label "drawB" draws two boards on each side.
   Line 524 calls the procedure "drawblocksC" to draw all the blocks.
   Line 525 calls the procedure "ballrun" to adjust the ball's movement.
   Line 527~540 determine if players missed to catch the ball and lose.
   If lost, initializes variables and turns to the losing interfaces.
   Line 541~553 determine if players hit all the blocks and win.
   If win, initializes variables and turns to the winning interfaces.
   Line 554 calls the procedure "stoppp" to slow down the program.
   Line 555 calls instruction in Irvine32 library and detect the input key.
   If there's an input, jump to label "L2" and calls procedure "listenerC".
   Line 502~564 will run as loop until program calls WIN/LOSE procedure.
   Line 565~566 will call back to the main.asm.

⇨ The game mode of "Teawork".

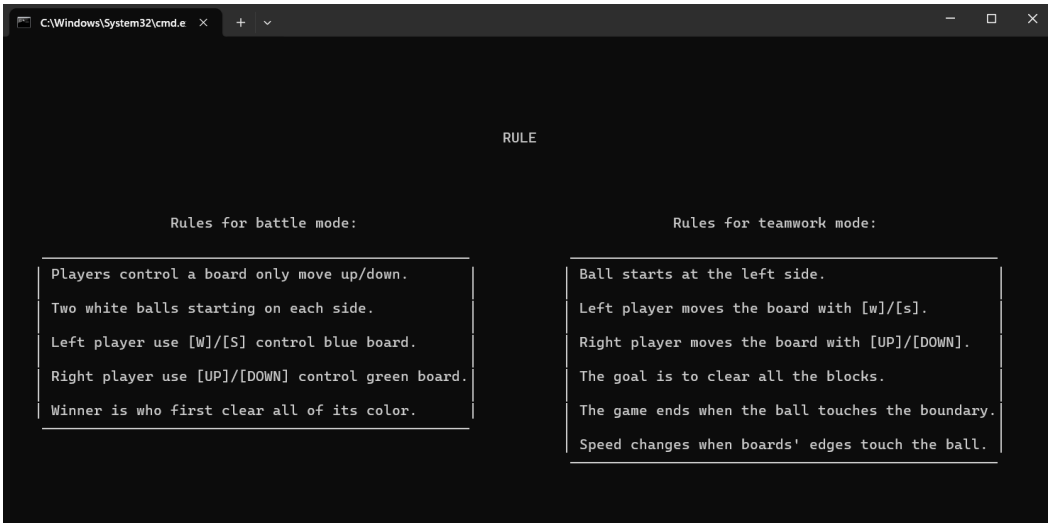
Initialize data or variables for players to replay the game again.

Screenshots of the Project

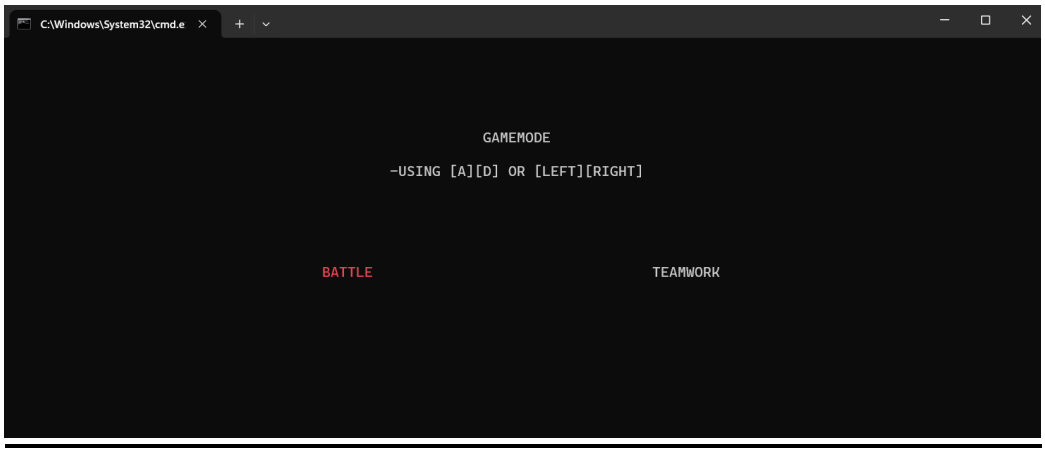
The main menu interfaces of the project



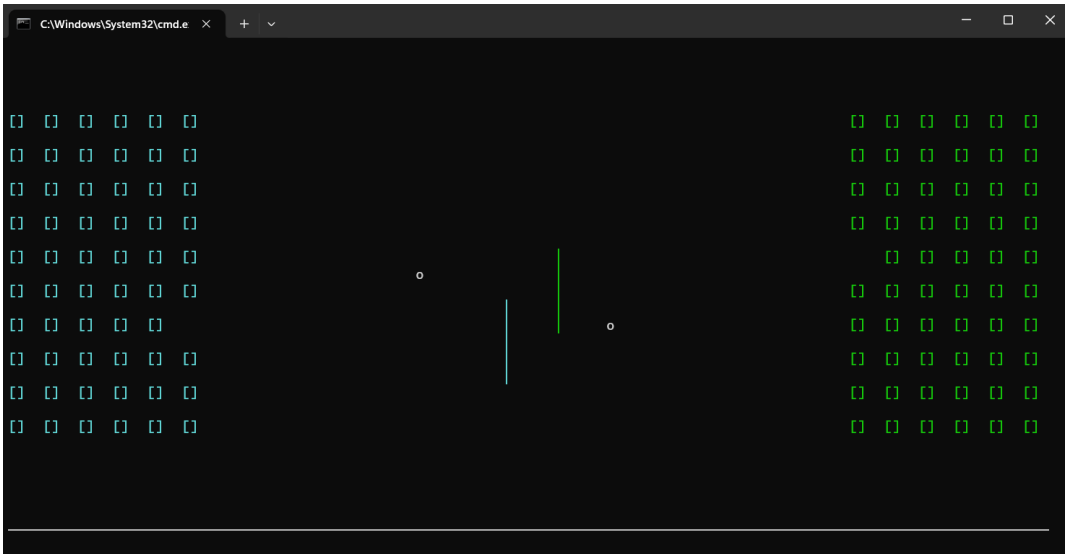
The rule interfaces of the project



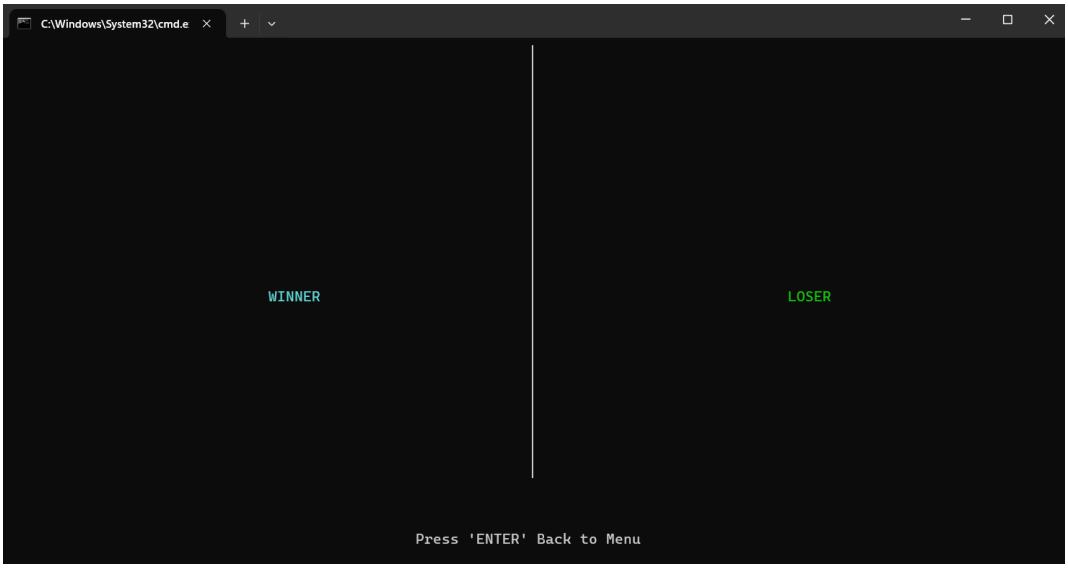
The setting interfaces of the project

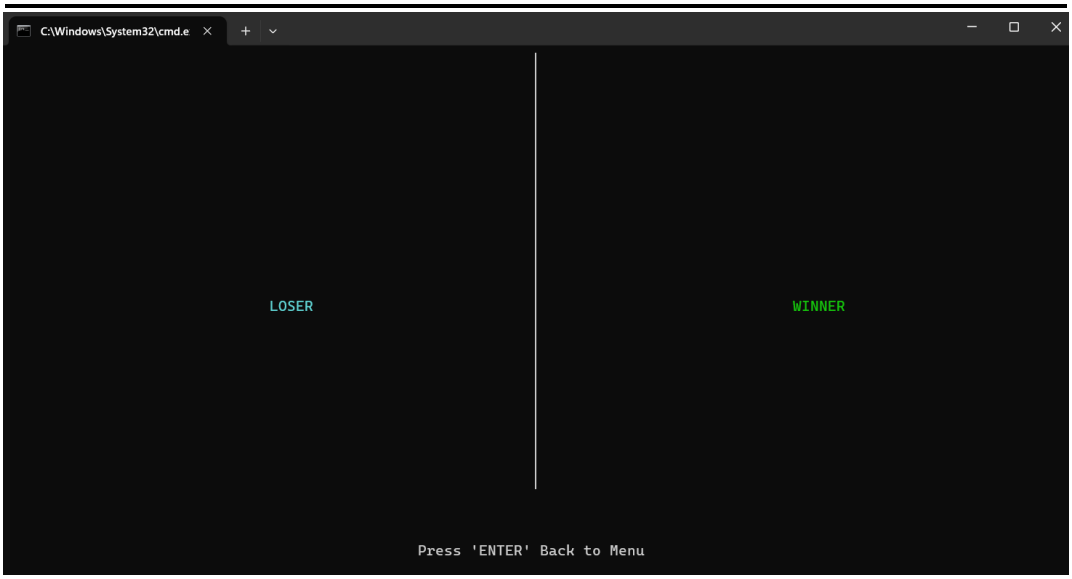


The game mode “battle” interfaces

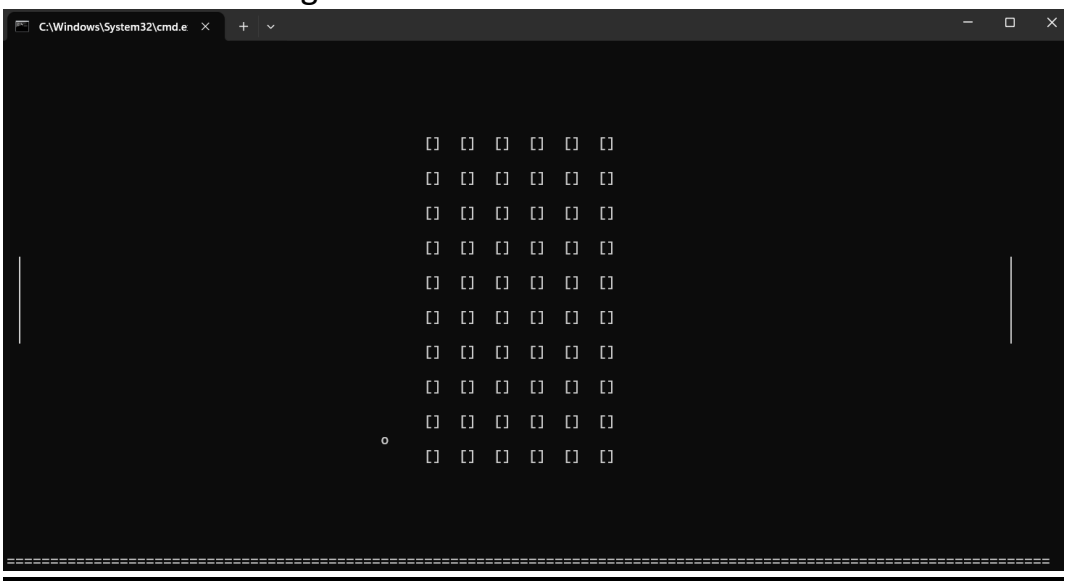


The ending interfaces of game mode “battle”

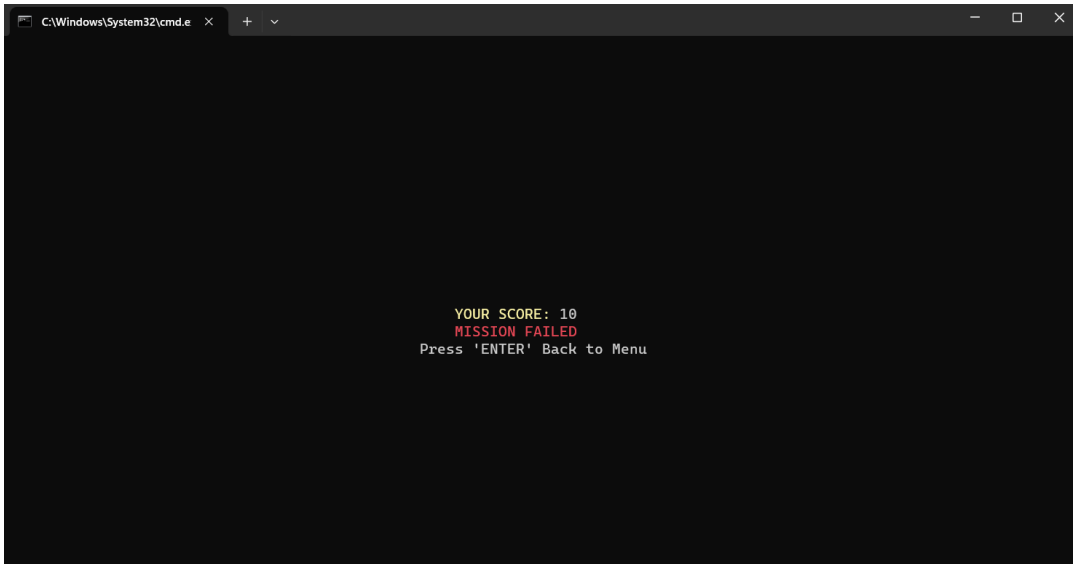


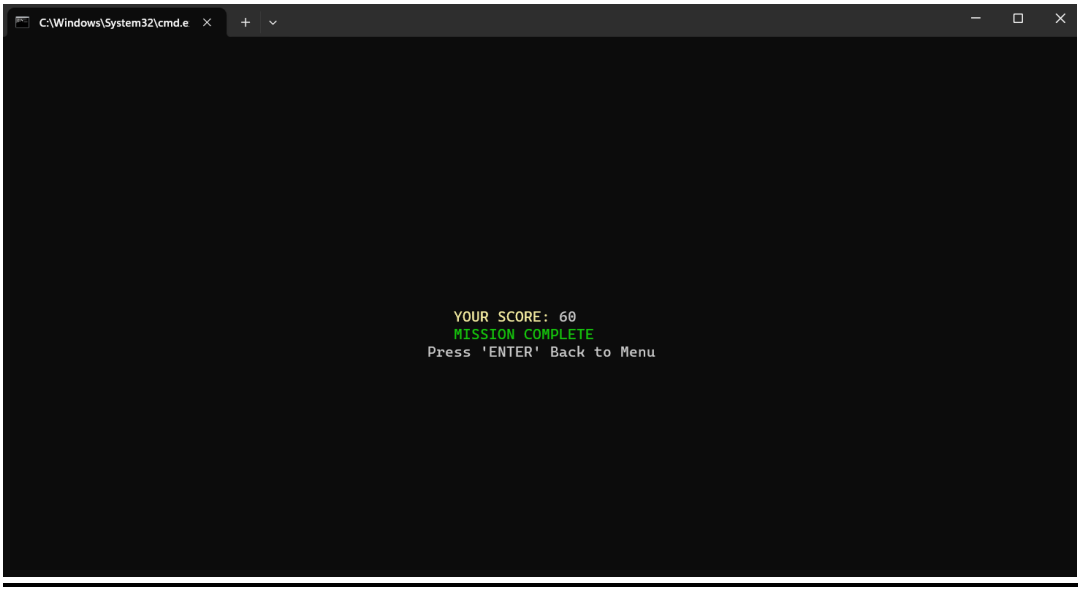


The game mode “teamwork” interfaces



The ending interfaces of game mode “teamwork”





Dilemmas and Solutions

D1: The boards and balls may fly out the screen and wherever they went, we couldn't figure out.

S1: Set the boundary for balls and boards, when balls touched the boundary then negative their speed to achieve the goal of rebound. As for the boards, when they touched the boundary then stop the movement to fixed the board there.

~~~~~~

D2: The movement of balls stayed in the regular path that wouldn't go another way to hit the blocks.

S2: Give some variety element into the game. That is, when balls' collision with the boards' edge, then rase up their speed to disturb the original path.

~~~~~~

D3: Visual Studio 2019 cannot support the environment of "int 21h" and other external models. What's more, the procedure "Readchar" included in the Irvine32 library taught in the course couldn't achieve the desired effect, which is caught the input key from the players at any time.

S3: Represent the function by using the procedure “Readkey”, which would set up the zero flag while it didn’t get any of the input key. In the other hand, by detecting zero flag, programs could know that whether there comes an input.

*However, it generated another problem that programs can only caught an input key at once. (Not a good solution for a two-players game)

~~~~~~

D4: How to determine that balls’ collision with the blocks or boards? How to present the destroyed blocks? And how to present the rebounds of the ball path?

S4: Check whether the position of balls at the next microsecond has blocks, whose value is 5Bh or 5Dh. If yes, turn the value of blocks to 20h, which represents blanks, and negative the balls’ speed to make it rebound. Same steps for checking the collision with the boards.

*Just to notice that there would be different speed while the ball hit the edge of the boards.

~~~~~~

D5: It couldn't well-showed out the balls' movement trajectory, because the updating of the screen is too frequently. Moreover, the screen would quickly flash again and again.

S5: Adding a delay procedure into the program, letting it runs millions of times of empty loops. Therefore, creating an illusion of delaying times.

~~~~~~

D6: The requirement of restarting the game. There's no better game than the game that could play again and again.

S6: Whenever called the final.asm or final1.asm, which represent two different game mode, it would initialize all the data and variables from the beginning of the game.

Reflection

111502509:

It's cool to write a game by assembly language. At the beginning, I never thought it would be a possible goal to achieve. Because things we learned in class seem too simple but then we worked together as a group and searched online. Based on some references, we figured out that the game breakout clone is possible to be built by assembly language. Although there are many things that can be improved but we certainly learned a lot from this final project. There is a major problem we can solve is that two players might press the keyboard on the same time which will confuse the computer and sometimes a key pressed maybe ignored by the computer in this condition. We tried to solve it and did some research but nothing worked. Hope we can learn to solve this problem in the future.
~~~~~~

111502510:

After I finished the course of the assembly language, I think I have learned a lot of skills of the realm. However, when I began to construct our own game by the skills learning from the class, it was really hard to achieve most function or feature about games. So, I surfed the internet to search for some solution of our game, just like how to read keyboards input without waiting the key press. Then also learned about more coding skills of the assembly! Although we didn't use more library to prettify the game's appearance, I think our game's playability is better than the original "breakout clone".

111502511:

It's a very enrich semester for making a project with a whole new computer language. Although our project's topic seemed no good enough, in order to rich the contents, we thought of creating it for more than just one player. There's an old saying that "When two people play together, than the joy not only doubles up". In addition to bring more pleasure to the game, we also designed two game modes for players to choose. I believe that it definitely is an innovative breakout clone game. Using assembly language without those high-level computer languages such as C++ or java, it really makes everything more difficult. Compare to the easily readable way, we have to program originally. It not only challenging our basic of operations of computer but also training us to be logically standardization. Through the process of final project, I deeply appreciate my growth of programing experience. And I can't wait to expect for the next issue letting me show off all my skills.

111502512:

I think that it's really a nice way finishing the course by making a project rather than doing paper exam as usual. We can't examine our learning effectiveness by only reading text book or PPT but doing projects, because we'll encounter some trouble that we couldn't think of during the time. When we're finding the solution, it's undoubtedly a huge improve to us. The game our group made was "Breakout clone", due to its widely known and easy to get started. However, we don't want it to be the same as normal, so we plugged in two different game mode to made it more stunning, which are battle mode and cooperative mode. It was truly an unforgettable experience to accomplish the project with others.