NCU 2022

Data Structure

Assignment #3

Shogi Game (Linked List)

鄭宇軒

通訊二 · 110503534

Abstract

This is a game project which simulates the gameplay of Shogi. It is written in C and uses the linked list data structure. It supports real-time matches and replay. Two users can play in the same match. All the steps will be recorded and can be outputted to a file. The save file, then, can be loaded for later replay.

Table of Contents

Features

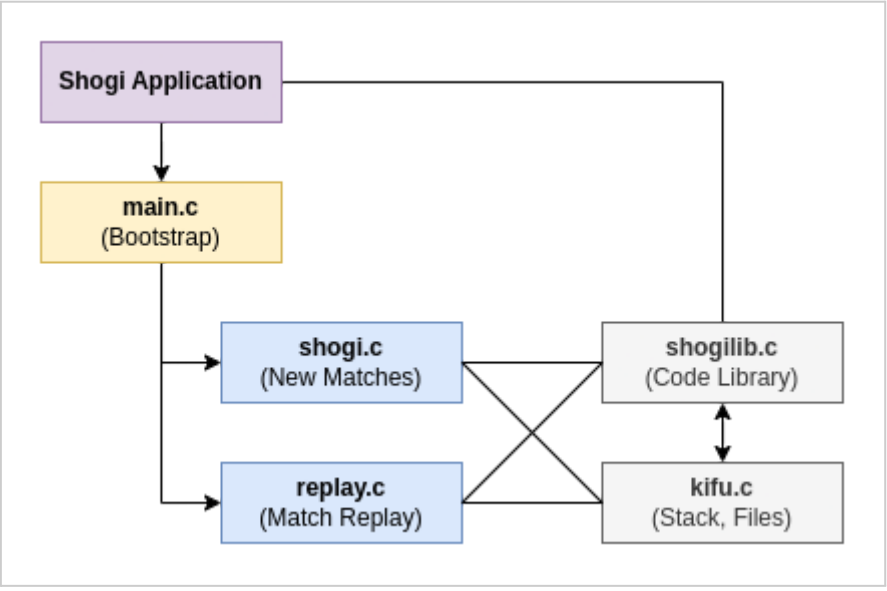
- Shogi game simulation
    - Different shogi pieces
    - Rendering of the kifu
    - Movement of pieces
    - Movement validation
    - “Eating” of pieces
- Reversion of steps
- Kifu record
    - File output of kifu
- Replay of a match (w/ timeline controls)
- Timer between moves in matches

Kifu Scope

Rather than recording everything on the entire board, this project records the actions of individual pieces. For reversion/replay, the system moves the pieces according to the action records.

There are two types of actions —— moving and “eating”. In the case of “eating”, the eaten piece (被吃棋子) will be recorded first, then the eater piece’s (吃棋方) movement will be recorded.

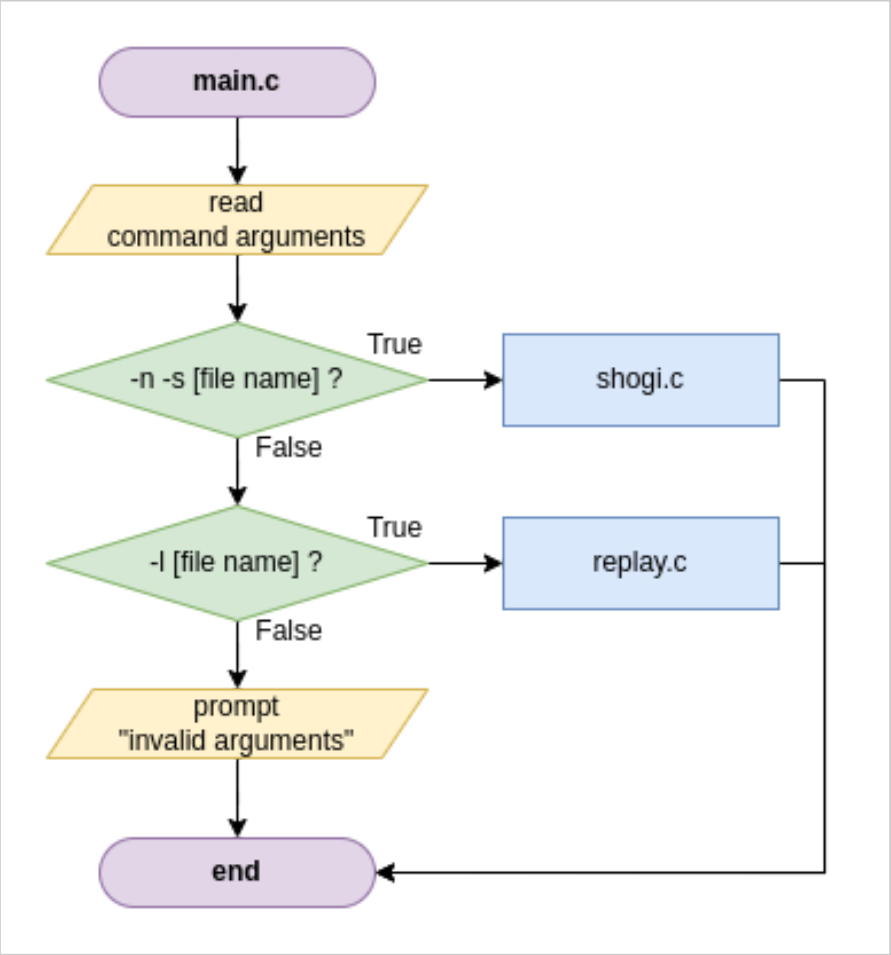
Program Structure



Modules

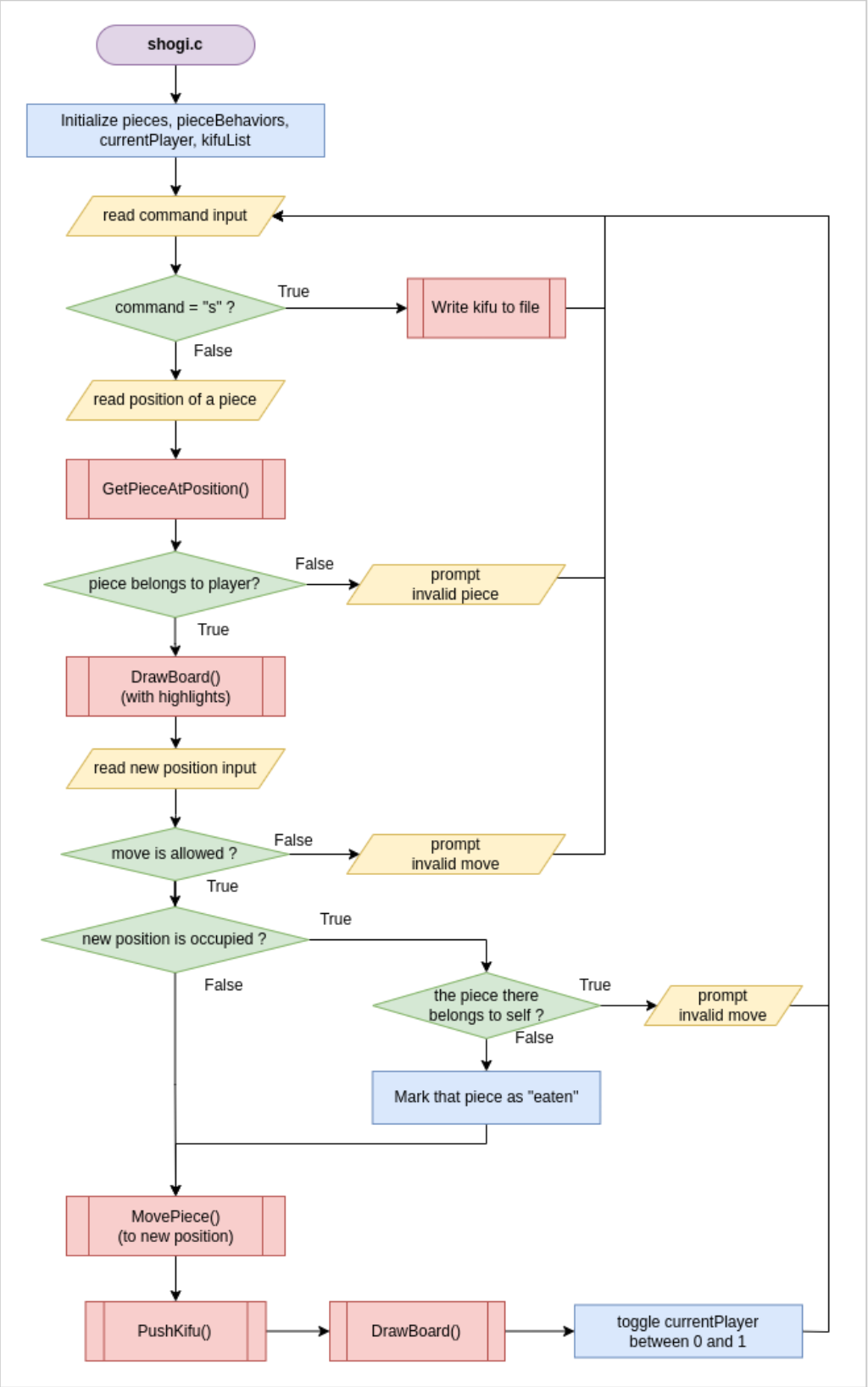
main.c

This is the entry point of the program. It will extract and compare the command arguments and launch the appropriate modules.



shogi.c

This module handles the real-time match of a shogi game. Two players can move the pieces in turns. It is responsible for parsing the user inputs and performing match actions (such as checking the moves, moving pieces, recording actions).

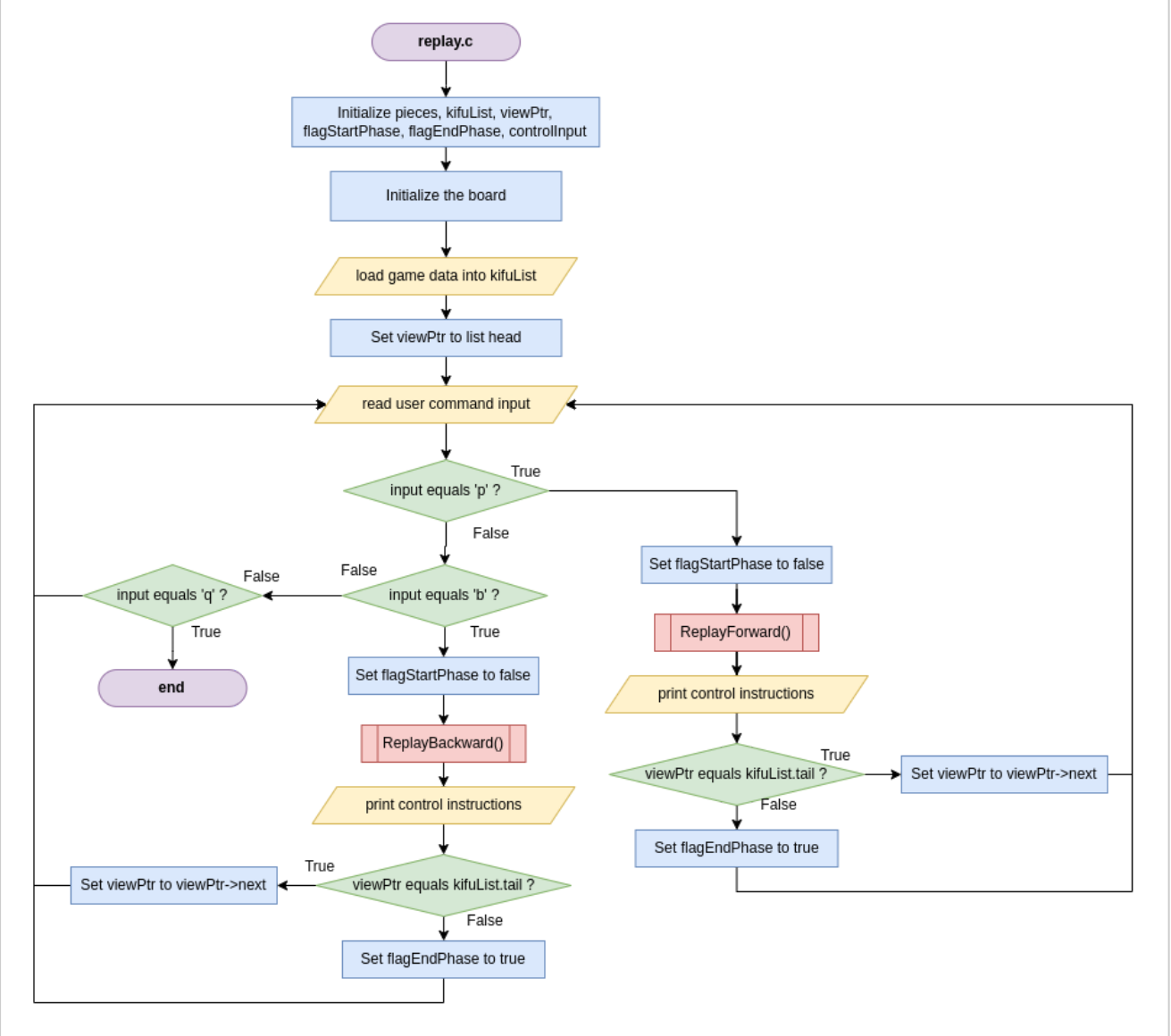


replay.c

This module handles the replay of a specific match. It will load a saved match from a file. The user can switch back and forth to see each step.

For “eating” actions, the eat and the move actions are consecutive. Therefore, the system will play one more step if the “eating” actions are involved.

viewPtr: *A dynamic pointer which moves along the kifu list*
flagStartPhase: *Is viewPtr at the head of kifuList?*
flagEndPhase: *Is viewPtr at the tail of kifuList?*



shogilib.c

This module includes the fundamental data types and methods required. It provides low-level functions like drawing the board and algorithms for validating the moves.

kifu.c

This module provides functions and data types for the kifu (match record). It includes a stack for recording the steps, as well as methods to read/write a kifu stack from/to a file.

The format of saving the kifu stack is as follows (headers are not included):

oldX	oldY	newX	newY	type	isEaten	Player
2	7	2	6	8	0	0
2	3	2	4	8	0	1
2	6	2	5	8	0	0
…						

Data Structure

`enum` PieceType

An enum containing all available piece types.

name	label
King	王
King2	玉
Rook	飛
Bishop	角
Gold	金
Silver	銀
Knight	桂
Lance	香
Pawn	歩

`struct` PieceBehavior

Contains the allowed behavior of a piece. Used for move validation.

name	type	description
type	PieceType	The type of the piece
n	int	Is the piece allowed to move in the specified direction?
ne	int	
e	int	
se	int	
s	int	
sw	int	
w	int	
nw	int	
allowMulti	int	Can the piece move multiple units at a time?
allowJump	int	Can the piece jump over other pieces?

struct Piece

Contains properties of a piece. It is dynamic.

name	type	description
type	PieceType	The type of the piece
player	int	Which player does this piece belong to (0 or 1)
x	int	The x-coordinate of this piece
y	int	The y-coordinate of this piece
isEaten	int	Is this piece being eaten?

struct KifuRecord

Records data of the movement of a piece.

name	type	description
type	PieceType	The type of the piece
oldX	int	The x-coordinate before moving
oldY	int	The y-coordinate before moving
newX	int	The x-coordinate after moving
newY	int	The y-coordinate after moving
player	int	Which player does this piece belong to (0 or 1)
isEaten	int	Is this piece being eaten?

struct Node

A linked list node.

name	type	description
record	KifuRecord	The type of the piece
prev	struct node*	Pointer to the previous item
next	struct node*	Pointer to the next item

struct NodePtr
A pointer to a link list node. Equivalent to **struct node* .**

struct KifuList
A linked list containing kifu data.

name	type	description
head	NodePtr	Pointer to the first item
tail	NodePtr	Pointer to the last item

Methods

shogi.c

void RequestInput(Pieces* pieces**, PieceBehavior*** pieceBehaviors**, int** currentPlayer**, KifuList*** kifuList**, const char*** kifuFileName**, unsigned** restartTimer**)**

Request game input from the user, and process the move.

Parameter	Description
Pieces* pieces	The array containing all 40 pieces
PieceBehavior* pieceBehaviors	The array containing behavior of all the pieces
int currentPlayer	Which player's turn is this (0 or 1)
KifuList* kifuList	The pointer to the kifu list
const char* kifuFileName	The name of the save file
unsigned restartTimer	Should the timer be restarted? (Used as boolean)

---

void ShogiGame(const char* kifuFileName**)**

The entry point of a new shogi match.

Parameter	Description
const char* kifuFileName	The name of the save file

shogilib.c

char* GetHans(PieceType type**)**

Get the Hans representation of a piece type.

Parameter	Description
PieceType type	The type of the piece

Returns: A Hans character representing the type of a piece

---

void DrawBoard(Piece* const pieces**, int** highlightX**, int** highlightY**)**

Render all the pieces and the board. Highlights a piece when applicable.

Parameter	Description
Pieces* pieces	The array containing all 40 pieces
int highlightX	The x-coordinate of the piece to highlight
int highlightY	The y-coordinate of the piece to highlight

---

void AssignPiece(Piece* piece**, int** player**, PieceType** piece**, int** x**, int** y**, int** isEaten**)**

Directly assign data to a piece. Used for initialization and replay.

Parameter	Description
Pieces* piece	Pointer to the piece to assign
int player	Which player does this piece belong to
PieceType type	The type of the piece
int x	The x-coordinate
int y	The y-coordinate
int isEaten	Is this piece being “eaten”?

`void InitializeBehaviors(PieceBehavior* pieceBehaviors)`

Builds an array of piece behavior. Contains predefined rules for all the pieces.

Parameter	Description
`PieceBehavior*` `pieceBehaviors`	The array of piece behaviors

`void InitializeBoard(Piece* const pieces)`

Initializes the starting position for all the pieces.

Parameter	Description
`Piece* const` `pieces`	The array of all the pieces

`Piece* GetPieceAtPosition(Piece* pieces, int x, int y)`

Get the pointer to a piece at the given position. Returns NULL if not found.

Parameter	Description
`Piece* const` `pieces`	The array of all the pieces
`int` `x`	The x-coordinate of the piece
`int` `y`	The y-coordinate of the piece

Returns: A pointer to the found piece

`PieceBehavior* GetPieceBehavior(PieceBehavior* pieceBehaviors, PieceType type)`

Get the behavior object for a specific piece type. Returns NULL if not found.

Parameter	Description
`PieceBehavior*` `pieceBehaviors`	The array of all the piece behaviors
`PieceType` `type`	The target piece type

Returns: A pointer to the found piece behavior

int CheckMoveRule(Piece* pieces**, PieceBehavior*** pieceBehaviors**, PieceType** type**, int** oldX**, int** oldY**, int** newX**, int** newY**, int** currentPlayer**)**

Check whether a move is valid.

Parameter	**Description**
Pieces* piece	Pointer to the piece to assign
PieceBehavior* pieceBehaviors	The array of all the piece behaviors
PieceType type	The type of the piece
int oldX	The x-coordinate before moving
int oldY	The y-coordinate before moving
int newX	The x-coordinate after moving
int newY	The y-coordinate after moving
int currentPlayer	Which player does this piece belong to

Returns: A boolean (int) indicating whether the move is valid

---

void MovePiece(Piece* pieces**, int** x**, int** y**)**

Move a piece to a new position.

Parameter	**Description**
Piece* const pieces	The array of all the pieces
int x	The target x-coordinate
int y	The target y-coordinate

kifu.c

`int IsListEmpty(KifuList* list)`

Check if a kifu stack is empty.

Parameter	Description
`KifuList* list`	Pointer to the kifu list

Returns: A boolean (int) indicating whether the list is empty

---

`void PushKifu(KifuList* list, const KifuRecord record)`

Push a kifu record to the kifu stack, if the stack is not full.

Parameter	Description
`KifuList* list`	Pointer to the kifu list
`const KifuRecord record`	The record to be pushed

`KifuStack* PopKifu(KifuList* const list)`

Get the last kifu record from a kifu list, if the list is not empty.

Parameter	Description
`KifuList* const list`	Pointer to the kifu list

Returns: A pointer to the kifu record

void WriteKifuToFile(const char* fileName, KifuList* const list)

Write a kifu list to a text file.

Parameter	Description
const char* fileName	The target file name
KifuList* const list	Pointer to the kifu list

---

void LoadKifuFromFile(const char* fileName, KifuList* const list)

Load a kifu stack from a text file.

Parameter	Description
const char* fileName	The target file name
KifuList* const list	Pointer to the kifu list

---

int CheckHasRevertibleSteps(KifuList* const list, const int currentPlayer)

Check if there are revertible steps for a specific player. Used for step reversion and replay.

Parameter	Description
KifuList* const list	Pointer to the kifu list
const int currentPlayer	The index of the player

Returns: A boolean (int) indicating if there are revertible steps

---

void PrintNode(NodePtr const node)

(Debug) prints the content of a list node.

Parameter	Description
NodePtr const node	Pointer to the target node

void PrintList(KifuList* const list**)**

(Debug) prints the content of a kifu list.

Parameter	Description
KifuList* const list	Pointer to the target kifu list

---

replay.c

void ShogiReplay(char* kifuFileName**)**

Load a kifu stack from a text file.

Parameter	Description
char* fileName	The target file name

Compiling

First, make sure the libev library is installed.

To compile the program, switch to the project directory. Use the following commands with gcc:

```
> gcc main.c -lm shogi.c -lm -lev shogilib.c -lm kifu.c -lm replay.c -lm
-o shogi.out
```

A new file `shogi.out` will be generated.

Running the Application

First, switch to the directory where the `shogi.out` file is located.

To start a new game:

```
> ./shogi.out -n -s {SAVEFILE NAME}
```

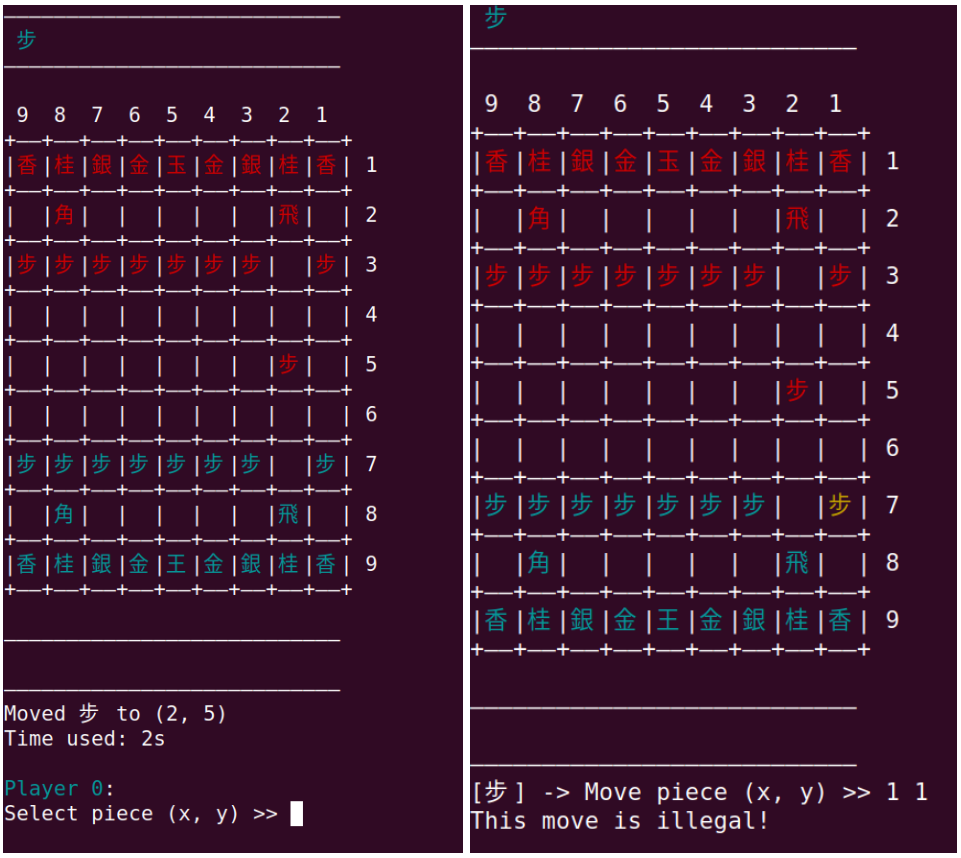
To replay an existing game:

```
> ./shogi.out -l {SAVEFILE NAME}
```

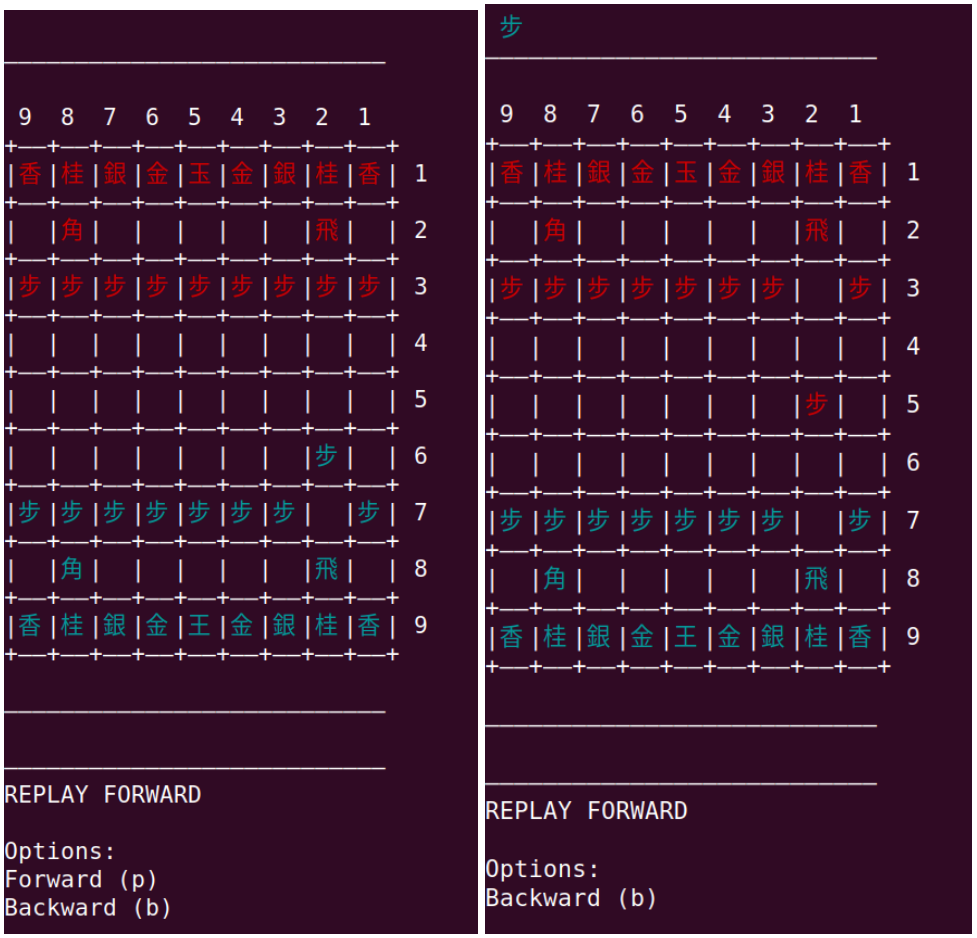
The arguments must be present, or the program will not run.

Demo Screenshots

New Game



Game Replay



Save File