Assignment 1 – MIS and MID

Designed by struct by_lightning{};
Group 6
Kuir Aguer, Brendan Duke, Jean Ferreira,
Zachariah Levine and Pranesh Satish

Module: board

Defines a 6 row by 7 column Connect Four board, as an abstract data type (ADT) Board. Contains access programs to interact with the board. When compiling the production application, use the compiler flag -DNDEBUG.

--- Interface ---

Uses:

None

Defined Macros/Constants:

NUM_ROWS: integer
The number of rows in the Connect Four board

NUM_COLS: integer
The number of columns in the Connect Four board

Defined Types:

Token: an enumeration with elements: EMPTY, RED, BLUE
The type of the elements that are contained in, inserted to, and returned from the ADT Board

Board: a pointer or reference to the board “object”
Variables of this type are instances of the ADT Board

Access programs:

board_create:
Return Type: Board
Parameters: None
Creates and returns a Board object, with all cells initialized to EMPTY. Terminates the program if the board cannot be created.

board_destroy:
Return Type: void
Parameters: Board b
Frees the memory allocated to the Board b.

board_checkCell:
Return Type: Token
Parameters: Board b, integer row, integer col
Returns the value of the board element in (row, col).

board_dropToken:
Return Type: integer
Parameters: Board b, Token token, integer col
Inserts the token, into the board b, in the column col. The inserted position is determined by where the token would fall in a physical Connect Four board, by gravity. Specifically, the token will be inserted into the position (row, col) such that the board b has the value EMPTY at (row, col) and row is as large as possible. The function returns 0 if the token was successfully inserted into the board and -1 if it wasn't successful (the column is already full or it was passed a NULL pointer for b).

board_dropPosition:
Return Type: integer
Parameters: Board b, integer col
Returns the row where a token be inserted by the board_dropToken (see board_dropToken). Returns -1 if the column is already fullor it was passed a NULL pointer for b.

board_empty:
Return Type: void
Parameters: Board b
Sets all cells of the board b to EMPTY.

--- Implementation ---

Uses:

board.h
stdio.h
stdlib.h
assert.h

Type Definitions/Structure,Union,Enumeration Declarations:

board_type: The internal representation of the board ADT; a structure of a two dimensional array of type Token. The top left of the board is defined to be (0,0).

```
struct board_type {
        Token board[NUM_ROWS][NUM_COLS];
};
```

Variables:

None

Access Programs:

Board board_create(void)

Creates an empty board by allocating memory of size the structure board_type.

```
Board b = (Board) malloc(sizeof(struct board_type));
board_empty(b);
return b;
```

void board_destroy(Board b)

Deallocates the memory associated with b.

```
free(b);
return;
```

Token board_checkCell(Board b, int row, int col)

Return the value of the board element at (row, col) by accessing the two-dimensional array in the structure that b points to.

```
assert(b != NULL);
if (b == NULL) return EMPTY;
return b->board[row][col];
```

int board_dropToken(Board b, Token token, int col)

Inserts the token, into the board b, in the column col. The inserted position is determined by where the token would fall in a physical Connect Four board, by gravity. Specifically, the token will be inserted into the position (row, col) such that the board b has the value EMPTY at (row, col) and row is as large as possible. The function returns 0 if the token was successfully inserted into the board and -1 if it wasn't successful (the column is already full or it was passed a NULL pointer for b).

```
assert(0 <= col && col < NUM_COLS);
if (b == NULL) return -1;
int row;

if ((row = board_dropPosition(b, col)) != -1) {
        b->board[row][col] = token;
        return 0;
}
return -1;
```

int board_dropPosition(Board b, int col)

Returns the row where a token be inserted by the board_dropToken (see board_dropToken). Returns -1 if the column is already full or it was passed a NULL pointer for b. This was done by looping down the column of the two-dimensional array as long as the next element in the column was empty.

```
assert(0 <= col && col < NUM_COLS);
if (b == NULL) return -1;
int row;

if (b->board[0][col] != EMPTY) return -1;
for (row = 0; row + 1 < NUM_ROWS && b->board[row + 1][col] == EMPTY; row++);
return row;
```

void board_empty(Board b)

Sets all cells of the board b to EMPTY by looping through every element in the two-dimensional array and setting its value to EMPTY.

```
if (b == NULL) return;
int row, col;

for (row = 0; row < NUM_ROWS; row++)
        for (col = 0; col < NUM_COLS; col++)
        b->board[row][col] = EMPTY;

return;
```

Module: linked list

A module that implements a generic linked list data structure.

--- Interface ---

Uses:

stdlib.h

Defined Macros/Constants:

None

Defined Types:

List<T>: A variable with only private members.
A variable of this type is a linked list.

Access programs:

addToList:

Return type: List<T> *
Parameters: T *newitem, List<T> *list
Add item T to the list to the front of the list.

deleteFromList:

Return Type: List<T> *
Parameters: T *toDeleteItem, List<T> *list
Delete the first occurrence of item T from the list.

traverseList:

Return Type: None
Parameters: void (*f)(T *item), List<T> *list
Parameters: void (*f)(T *item), float dt, List<T> *list
Iterate over the list and execute function f on each node of the list.
Note that this is a overloaded function which can use either set of parameters

reduceList:

Return type: T *
Parameters: bool (*f)(T *listItem, T *item), T *newest, List<T> *list
Iterate over the list and return the first item T that satisfies the function f

--- Implementation ---

Uses:

stdlib.h

Type Definitions/Structure,Union,Enumeration Declarations:

List: The internal representation of a linked list; a class representing our linked list node and containing the functions that can modify the list.

```
class List {
        T *item;
        List<T> *next;

        // private constructor so that clients can't make a LIst
        // object except by using addToList(item, null);
        List() {}
}
```

Variables:

None

Access Programs:

template<typename T> List<T> * List<T>::addToList(T *newItem, List<T> *list)
Adds item T to the front of the linked list, list

```
if(newItem != NULL) {
   List<T> *resultList = (List<T> *)malloc(sizeof(List<T>));
   resultList->item = newItem;
   resultList->next = list;
   return resultList;
 }
 else {
   return list;
 }
```

template<typename T> List<T> * List<T>::deleteFromList(T *toDeleteItem, List<T> *list)

Delete the first occurrence of item T from the list, list.

```
if(list != NULL) {
  List<T> *current;
  List<T> *previous;
  for(current = list, previous = NULL;
      current != NULL && current->item != toDeleteItem;
      previous = current, current = current->next);
  if(current != NULL) {
    if(previous != NULL) {
      previous->next = current->next;
    }
    else {
      list = list->next;
    }
    free(current);
  }
  return list;
}
else {
  return NULL;
}
```

template<typename T> void List<T>::traverseList(void (*f)(T *item), List<T> *list)
template<typename T> void List<T>::traverseList(void (*f)(T *item, float dt), float dt, List<T> *list)

Iterate over the list and execute function f on each node of the list.
Note that this is a overloaded function which can use either set of parameters

```
for(List<T> *current = list;
    current != NULL;
    current = current->next) {
  (*f)(current->item);
}
```

template<typename T> T * List<T>::reduceList(bool (*f)(T *listItem, T *item), T *newest, List<T> *list)

Iterate over the list and return the first item T that satisfies the function f

```
 for(; list != NULL; list = list->next) {
    if((*f)(list->item, newest) == true ) {
    return list->item;
  }
 }
 return NULL;
```

Module: graphics

Responsible for all graphical representation on the screen, such as loading media files, setting up coordinates for the game layout, and positioning images throughout the screen accordingly.

--- Interface ---

Uses:

SDL2/SDL.h
string
board.h
linkedList.h

Constants:

CONNECT4_WINDOW_OFFSET_Y: int
Prevents the application window to open beyond the boundaries of the client's screen
SCALE: float
Allows the game screen to be scaled
SCREEN_WIDTH: int
Window width size for the game
SCREEN_HEIGHT: int
Window height size for the game
TOKEN_WIDTH: int
Width of the tokens used for the game
TOKEN_HEIGHT: int
Height of the tokens used for the game
GRID_OFFSET_Y: int
Offset of column lines for the game playing board
GRID_OFFSET_X: int
Offset of row lines for the game playing board
GRID_WIDTH: int
Width of the game board
GRID_HEIGHT: int
Height of the game board
MAINMENU_SETUP_BUTTON_LEFT: int
Calculates the left position the setup button
MAINMENU_SETUP_BUTTON_RIGHT: int
Calculates the right position the setup button
MAINMENU_SETUP_BUTTON_TOP: int
Calculates the top position the setup button
MAINMENU_SETUP_BUTTON_BOTTOM: int
Calculates the bottom position the setup button
MAINMENU_QUIT_BUTTON_LEFT: int
Calculates the left position the quit button
MAINMENU_QUIT_BUTTON_RIGHT: int
Calculates the right position the quit button

MAINMENU_QUIT_BUTTON_TOP: int
Calculates the top position the quit button
MAINMENU_QUIT_BUTTON_BOTTOM: int
Calculates the bottom position the quit button
SETUP_BOTTOM_BUTTONS_OFFSET: int
Calculates the bottom position of all the buttons in the bottom of the setup game mode screen
SETUP_CLICKY_TOKENS_OFFSET: int
Calculates the position of the button that changes the current token colour in play(ie. Going from blue's turn to red's turn and vice versa)
SETUP_2PLAYER_BUTTON_WIDTH: int
Calculates the width of the '2 player' game mode button within the setup screen
SETUP_2PLAYER_BUTTON_HEIGHT: int
Calculates the height of the '2 player' game mode button within the setup screen
SETUP_1PLAYER_BUTTON_WIDTH: int
Calculates the width of the '1 player' game mode button within the setup screen
SETUP_1PLAYER_BUTTON_HEIGHT: int
Calculates the height of the '1 player' game mode button within the setup screen
SETUP_MENU_BUTTON_WIDTH: int
Calculates the width of the 'menu" button within the setup screen
SETUP_MENU_BUTTON_HEIGHT: int
Calculates the height of the 'menu" button within the setup screen

Global Declarations:

extern List<FallingToken> *gFallingTokens
A list that represents all the falling tokens during gameplay

Defined Structures:

FallingToken
Holds information about each token, which will help gravity to be simulated on each token individually.
TextureWrapper
Provides a way to store dimensions to a texture.
TokenLocation (incomplete type)
Allows a tokens to be defined by its color and position within the game board.
(the structure definition for this struct is visible in the interface)

```
struct TokenLocation {
   int row;
   int column;
  Token colour;
};
```

Access programs:

drawFallingToken:
Return Type: void
Parameters: FallingToken *token
Given a token, determine the position to drop it.

clearFallingToken:
Return Type: void
Parameters: FallingToken *fallingToken
Finds the position of the falling token.

updateFallingToken:
Return Type: void
Parameters: FallingToken *fallingToken, float dt
*Updates the position of a falling token depending on the time (*dt*) it has been airborne.*

mainMenuRender:
Return Type: void
Parameters: none
Rendering for the main menu

transitionSetupRender:
Return Type: void
Parameters: none
Positions and renders the buttons to be used in the setup game mode to allow players to either start a game from setup mode, or return to main menu.

setupRender:
Return Type: void
Parameters: none
Rendering for the setup game mode

init:
Return Type: boolean value
Parameters: none
Returns true if the program has been initialized and window has been created successfully.

loadMedia:
Return Type: boolean value
Parameters: none
Returns true if all media (images) required for the game has been successfully accessed.

close_sdl:
Return Type: void
Parameters: none
Ends the program properly and closes the window

dropToken:
Return Type: boolean
Parameters: Board b, Token tokenColour, integer col
Returns true if a token and been successfully dropped onto the given board, at the specified column.

deleteStillToken:
Return Type: void
Parameters: Falling Token *fallingToken
If falling token has reached the lowest possible position, stop the gravity simulation on that token.

setHighlightedTokenList:
Return Type: void
Parameters: List<TokenLocation> *highlightedTokenList
List of token location that are highlighted in the setup game mode.

--- Implementation ---

Uses:

SDL2/SDL.h
string
board.h
linkedList.h
graphics.h
stdlib.h

Type Declarations:

TextureWrapper:
Any object of this struct will be able to use SDL's texture function, and make it easier to calculate positioning for these textures by the use of the width and height.

```
struct TextureWrapper {
   SDL_Texture *texture;
   int width;
   int height;
};
```

FallingToken:
Allows manipulation of multiple falling tokens at the same time, independently of each other.

```
struct FallingToken {
    int x;         //distance from left to right
    int y;         //distance from top to bottom
    int v;          // velocity
    int yFinal;   // final position
    bool isFalling;
    Token token;  // enum {BLUE, RED, EMPTY}
};
```

Global Variables

*gFallingTokens: List<FallingToken>
This list keeps track of all the tokens that are falling in the game.

Variables

(Any variables beginning with a g (ie gVarName) is a global variable without external linkage (meaning its internal to module))

gRenderHighlighted: bool
Flag for tracking whether token highlighting should be rendered or not.

*gWindow: SDL_Window
Represents the window to be used for the game, any change with the window itself (height, width, border, and full screen) will be made through this.

*gConnect4Board: TextureWrapper
Represents the image of the game board.

*gRedToken: TextureWrapper
Represents the image of the red token.

*gBlueToken: TextureWrapper
Represents the image of the blue token.

*gMainMenu: TextureWrapper
Represents the image of the main menu.

*gOnePlayerButton : TextureWrapper
Represents the one player button

*gTwoPlayerButton: TextureWrapper
Represents the two player image button.

*gMenuButton: TextureWrapper
Represents the menu image button.

*gGlow: TextureWrapper
Token highlighting texture.

*gInvalidMessage: TextureWrapper
Invalid board message display

*gInvalidTokenMessage: TextureWrapper
Invalid game setup message display.

*gRenderer: SDL_Rendered
Responsible for rendering any image onto an SDL_Window.

*gHighlightedTokens: List<TokenLocation>
This list contains all currently highlighted tokens in game.

Local Programs:

static void highlightToken(TokenLocation *tokenToHighlight)

Given a tokenLocation pointer as argument, this function find the row and column of the respective token and highlights it. The TextureWrapper gGlow is used to highlight the tokens (it's simply a white disk with low alpha value that allows tokens to look 'highlighted').

```
static void highlightToken(TokenLocation *tokenToHighlight) {

   //find location of token
   SDL_Rect fillRect = {GRID_OFFSET_X +
            TOKEN_WIDTH*tokenToHighlight->column,
                  GRID_OFFSET_Y + TOKEN_HEIGHT * tokenToHighlight->row,
               TOKEN_WIDTH, TOKEN_HEIGHT};

   TextureWrapper *tokenColour =
               (tokenToHighlight->colour == RED) ?  gRedToken : gBlueToken;

   SDL_RenderCopy(gRenderer, tokenColour->texture, NULL, &fillRect);

   SDL_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0x66);
   SDL_SetRenderDrawBlendMode(gRenderer, SDL_BLENDMODE_BLEND);

   SDL_RenderFillRect(gRenderer, &fillRect);
   SDL_SetRenderDrawColor( gRenderer, 128, 128, 128, 0xFF );
   SDL_SetRenderDrawBlendMode(gRenderer, SDL_BLENDMODE_NONE);
   displayBoard();   // get the board ready

   // render the glow texture wrapper (gGlow) to the back buffer of the window.
   SDL_RenderCopy(gRenderer, gGlow->texture, NULL, &fillRect);
}
```

static void freeTexture(TextureWrapper *myTexture)

Responsible for de-allocating any Texture Wrapper. Given the argument of a pointer to TextureWrapper, destroy the texture with SDL_DestroyTexture, and set it to NULL.

```
static void freeTexture(TextureWrapper *myTexture) {
  if(myTexture != NULL) {
    if(myTexture->texture != NULL) {
      SDL_DestroyTexture(myTexture->texture);    //destroy texture
      myTexture->texture = NULL;
    }
    free(myTexture);   //built in to C standard library, free() releases from memeory
  }
}
```

static TextureWrapper* loadTexture(std::string path)

Load a media file from the given directory path and transform it into a texture, by using SDLs built in function as describe below

```
static TextureWrapper *loadTexture(std::string path) {

        Validate the path given (call SDL_GetError with unsuccessful)
        Create a new surface with SDL_CreateRGBSurface based on the scale   size of
        the game
        Create a new texture (newTexture) from SDL_CreateTextureFromSurface
        Allocate memory for newTexture
        Remove the surface created earlier, since its no longer needed, this can   be
        done with SLD_FreeSurface
        Return newTexture
}
```

static bool compareXPosition(FallingToken *listItem, FallingToken *item)

Returns true if the two tokens are in the same column, false otherwise

```
static bool compareXPosition(FallingToken *listItem, FallingToken *item) {
  return listItem->x == item->x;
}
```

static void freeTokenLocation(TokenLocation *tokenLocation)

Calls the standard library free() function, while removes the pointer of TokenLocation from memory.

```
static void freeTokenLocation(TokenLocation *tokenLocation) {
  free(tokenLocation);
}
```

Access Programs:

void mainMenuRender()

Renders the main menu background texture on the screen by calling local function displayMainMenu(). DisplayMainMenu() just copies a global texture gMainMenu onto gRenderer. MainMenuRender() also presents the renderer (causing the texture to be displayed on the screen).

```
void mainMenuRender() {
  displayMainMenu();
  SDL_RenderPresent(gRenderer);
}
```

void drawFallingToken(FallingToken *fallingToken)

The function creates a new tokenWrapper variable from the color, and position (x, y) of the falling token pointer passed in as argument.
This allows us to render the token onto the screen with SDL_RenderCopy (SDL's built-in rendering function).

```
void drawFallingToken(FallingToken *fallingToken) {
  TextureWrapper *tokenTexture;
  if (fallingToken->token == RED) {
    tokenTexture = gRedToken;
  }
  else {
    tokenTexture = gBlueToken;
  }

  SDL_Rect tokenRect;
  tokenRect.x = fallingToken->x;
  tokenRect.y = fallingToken->y;
  tokenRect.w = TOKEN_WIDTH;
  tokenRect.h = TOKEN_HEIGHT;

  SDL_RenderCopy( gRenderer, tokenTexture->texture, NULL, &tokenRect );
}
```

bool init()

Initializes all of SDL's graphical functions. Also creates a new window (gWindow) and renderer for the window (gRenderer). If everything is initialized successfully, it will return true.
If any initialization is unsuccessful the program will not start and will prompt the user with a message of what has gone wrong.

```
bool init() {

  bool success = true;

  if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    printf("SDL could not initialize! SDL_Error: %s\n", SDL_GetError());
  }
  else {
    // create window
    gWindow = SDL_CreateWindow("Connect 4", SDL_WINDOWPOS_UNDEFINED,
        SDL_WINDOWPOS_UNDEFINED, SCREEN_WIDTH, SCREEN_HEIGHT, 0);
    if(gWindow == NULL) {
      printf("Window could not be created! SDL_Error: %s\n",
          SDL_GetError());
    }
    else {
      //Create renderer for window
      gRenderer = SDL_CreateRenderer( gWindow, -1,
          SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC );
      if( gRenderer == NULL ) {
        printf( "Renderer could not be created! SDL Error: %s\n",
            SDL_GetError() );
        success = false;
      }
      else {
        //Initialize renderer color
        SDL_SetRenderDrawColor( gRenderer, 128, 128, 128, 0xFF );
      }
    }
  }
  return success;
}
```

bool loadMedia()

Any media (images, sounds, effects) must be declared in this function. If at least one media file is not loaded properly, the program will not start.
All images loaded are variables of the textureWrapper structure, which gives the image an SDL texture, a width and a height.

```
bool loadMedia() {

  bool success = true;

  gBackground = loadTexture("../misc/white_background.bmp");    // background image
  if (gBackground == NULL) {
    printf("Failed to load background!\n");
    success = false;
  }
   // code omitted: repeat the above 4 lines for any other media that requires to be
loaded
   // Make sure a global texture has been declared in order to be properly assigned to
the
   // media

  return success;
 }
```

void close_sdl()

When the program is closed, we must de-allocate all textureWrapper surface variables. To do so we first destroy the texture by using freeTexture, which in turn use SDL_DestroyTexture. Once the texture is destroyed we can safely set the variables to NULL. Next all windows must be destroyed and set to NULL, which can be done with SDL_DestroyWindow. Finally SDL_Quit can be called, which will terminate SDL safely.

```
void close_sdl() {
  // for all the global texture variables we apply the same set of function calls
 // to de-allocate  and destroy each texture

  freeTexture(gConnect4Board);
  gConnect4Board = NULL;
  // code omitted: repeat above two steps for all other global textures

  // Destroy window
  SDL_DestroyWindow(gWindow);
  gWindow = NULL;

  // Quit SDL subsystems
  SDL_Quit();
}
```

void deleteStillToken(FallingToken *fallingToken)

If a token is no longer falling (isFalling == false) then we can remove it from the list gFallingTokens. The deleteFromList function, will return a new list of FallingToken pointers, and set it as the new gFallingToken list.

```
void deleteStillToken(FallingToken *fallingToken) {
  if(fallingToken->isFalling == false) {
    gFallingTokens = List<FallingToken>::deleteFromList(fallingToken, gFallingTokens);
  }
}
```

void transitionSetupRender()

This function is the transition state from MAINMENU to SETUP. TransitionSetupRender() clears the background, displays the two setup tokens used to choose whether blue or red tokens are dropped in SETUP mode, and renders the Menu, One Player and Two Player buttons on the screen. These are all textures that need to be rendered only once, during the transition from MAINMENU to SETUP, and not in the SETUP state itself.

void setupRender()

This function does the rendering for the SETUP state. It displays the board, then renders highlighted tokens if they have not been rendered since the last press of the "Two Player" button, which is determined by checking a boolean value " gRenderHighlighted" that is local to the graphics module. SetupRender also presents draws the texture stored in gRenderer onto the screen (it draws the state of the game in SETUP).

void clearFallingToken(FallingToken *fallingToken)

This function just overwrites a fallingToken's previous position with the background. This is to save rendering the whole background each frame (we just erase where the token WAS before it dropped a frame's distance further).

void updateFallingTokens(FallingToken *fallingToken, float dt)

This function updates the position/velocity of fallingToken based on a macro-defined constant acceleration, and an input time-step "dt". The function also implements bouncing and damping: when a fallingToken reaches its slot in the board it bounces upward and its speed is reduced until its speed is below a certain value. When a fallingToken's speed is below this epsilon value it is marked to be deleted used a bool value "isFalling" in the fallingToken struct.

bool dropToken(Board b, Token tokenColour, int col)

Before dropping the token in a column, we must first check that the column is not full. If the drop is allowed to be made, create a new FallingToken pointer, and declare all the values required by the FallingToken struct. Then the falling token can then be inserted in the list gFallingTokens, where it can be accessed for gravity simulation.

```
bool dropToken(Board b, Token tokenColour, int col) {

  // Find the row where the token should land, and check that it is not full
  int row = board_dropPosition(b, col);
  if (row == -1) {
    return false;
  }

  TextureWrapper *token;
  if (tokenColour == RED) {
    token = gRedToken;
  }
  else if (tokenColour == BLUE) {
    token = gBlueToken;
  }

  FallingToken *newToken = (FallingToken *)malloc(sizeof(FallingToken));

  // Initial position of the token
  newToken->x = GRID_OFFSET_X + TOKEN_WIDTH * col;
  newToken->y = GRID_OFFSET_Y;
  // check if there is another falling token that is above
  // the top of the board; if so drop this next token ABOVE that token
  FallingToken *currentHighest =
    List<FallingToken>::reduceList(compareXPosition, newToken, gFallingTokens);
  if(currentHighest != NULL) {
    if(newToken->y + TOKEN_HEIGHT > currentHighest->y) {
      newToken->y = currentHighest->y - TOKEN_HEIGHT;
    }
  }

  // Velocity of token
  newToken->v = 0;
  // Final height of the token
  newToken->yFinal = GRID_OFFSET_Y + row * TOKEN_HEIGHT;
  newToken->isFalling = true;
  newToken->token = tokenColour;

  gFallingTokens = List<FallingToken>::addToList(newToken, gFallingTokens);
  return true;
}
```

void setHighlightedTokenList(List<TokenLocation> *highlightedTokenList)

This function takes a List of TokenLocations (row, column and colour) and turns the data structure gHighlightedTokens into that List by first free'ing the old list, then setting gHighlightedTokens to the head of highlightedTokenList. This function also sets the global variable gRenderHighlighted to indicate to setupRender that the highlighted tokens must be rendered on the next frame flip.

void displayBoard()

Determine position of where the board should be located, and render the texture wrapper of the board (gConnect4Board) in the back buffer of the window with SDL_RenderCopy.

```
void displayBoard() {
  // determine the position for the board
  SDL_Rect DestR;
  DestR.x = GRID_OFFSET_X - 1;
  DestR.y = GRID_OFFSET_Y - 1;
  DestR.w = gConnect4Board->width;
  DestR.h = gConnect4Board->height;
  SDL_RenderCopy( gRenderer, gConnect4Board->texture, NULL, &DestR );
}
```

void displaySetupTokens()

Determine the position of the blue token and red tokens that are used to switch the colors in game, and render the texture wrapper of each token (gRedToken & gBlueToken) in the back buffer of the window with SDL_RenderCopy.

```
void displaySetupTokens() {
  // determine the position for the setup tokens
  SDL_Rect tokenRect;
  tokenRect.x = SETUP_CLICKY_TOKENS_OFFSET;
  tokenRect.y = GRID_OFFSET_Y;
  tokenRect.w = TOKEN_WIDTH;
  tokenRect.h = TOKEN_HEIGHT;

  //Render texture to screen
  SDL_RenderCopy( gRenderer, gRedToken->texture, NULL, &tokenRect );

  tokenRect.x = SCREEN_WIDTH - SETUP_BOTTOM_BUTTONS_OFFSET -
TOKEN_WIDTH;
  tokenRect.y = GRID_OFFSET_Y;
  tokenRect.w = TOKEN_WIDTH;
  tokenRect.h = TOKEN_HEIGHT;

  //Render texture to screen
  SDL_RenderCopy( gRenderer, gBlueToken->texture, NULL, &tokenRect );
}
```

void displayMainMenu()

Render the main menu texture wrapper (gMainMenu) to the back buffer of the window with SDL_RenderCopy.

```
void displayMainMenu(void)
{
  SDL_RenderCopy(gRenderer, gMainMenu->texture, NULL, NULL);
}
```

Module: sdl2_connect4

Contains the main game loop to run the Connect 4 program. All event handling from SDL_Events are done in the sdl2_connect4 module (e.g. mouse clicks).

--- Interface ---

Uses:

None

Defined Macros/Constants:

None

Defined Types:

None

Access programs:

connect4:
Return Type: integer
Parameters: none
Returns 0 for success. Plans to potentially return values other than 0 for different error codes. Runs the Connect 4 game.

--- Implementation ---

Uses:

board.h
graphics.h
gameLogic.h
stdio.h (C runtime library)
SDL.h (secret hidden by this module – event handling/graphics library)

Type Definitions/Structure,Union,Enumeration Declarations:

None

Variables:

The three arrays:
void (*handleEvents[NUMBER_OF_STATES])(GameState *gameState)
void (*logic[NUMBER_OF_STATES])()
void (*render[NUMBER_OF_STATES])()

are array variables hidden in the sdl2_connect4 implementation, which contain function pointers returning type void and taking one argument of type GameState, returning void and taking no arguments, and returning void and taking no arguments for the handleEvents, logic[] and render[] arrays, respectively. These arrays contain the different functionality that should happen when the game is in a different state. For example, if the current state were MAINMENU, then the render[] array would be indexed to a function that renders the main menu background.

Access Programs:

static void logicStub() {}
static void handleEventsStub(GameState *gameState) {}
static void renderStub() {}

These three function are empty stub functions, used in the handleEvents[], logic[] and render[] arrays when we are in a state that doesn't, for example, do any logic. E.g. the MAINMENU state indexes the logic[] array to logicStub().

static MenuState handleMainMenuMouseClick(int x, int y)

This function takes a point (an x, y pair) and checks if the point falls within the SETUP button region, in which case it returns the SETUP MenuState.

E.g.

```
if ((x >= MAINMENU_SETUP_BUTTON_LEFT) &&
    (y >= MAINMENU_SETUP_BUTTON_TOP) &&
    (x <= MAINMENU_SETUP_BUTTON_RIGHT) &&
    (y <= MAINMENU_SETUP_BUTTON_BOTTOM)) {
      return SETUP;
  }
```

Likewise, a click within the QUIT button region returns the QUIT MenuState. Otherwise, if the click is outside any buttons, the MAINMENU MenuState is returned.

static void mainMenuHandleEvents(GameState *gameState)

Displays and handles mouse clicks/motions when in the MAINMENU state. Transitions to SETUP state if mouse clicks are in the SETUP button region, by calling transitionSetupRender() in the graphics module. Sets game state to QUIT if mouse click is in QUIT button region. Mouse-clicks are handled by the handleMainMenuMouseClick() function.

void setupHandleEvents(Board b, int row, int col)

This function handle mouse clicks in the SETUP state. Events are pulled off the event queue using SDL_PollEvent. If an SDL_QUIT event is pulled off the event queue, the game state is set to quit, i.e.

```
if(e.type == SDL_QUIT) {
  gameState->currentState = QUIT;
}
```

Also, as in mainMenuHandleEvents, this function checks handles mouse-clicks. If a mouse-click is in the Two Player or One Player button regions, transitionSetupTwoPlayer(gameState) is called, which checks and reports any errors that occurred in the current SETUP of the board (e.g. if red won, or there is a difference > 1 between number of blue tokens and number of red tokens). If the game is in progress, then currently setupHandleEvents just prints a successful message to console.

If clicks are within the setup token radii then the game state's “currentToken” data member is set to RED or BLUE, depending on which setup token was clicked.
If the click is within the board, a token of the same colour as gameState->currentToken is dropped using the graphics module function dropToken(gameState->board, gameState->currentToken, dropColumn).

int connect4()

This function contains the game loop, which actually runs the entire game, controlling timing and calling all event handling functions, as well as physics-updating (logic) functions and rendering functions (graphics) in other modules. First the graphics are initialized by calling init() and loadMedia() from the graphics module. Then a GameState object, which contains the Board, current token colour, and MenuState (e.g. SETUP or QUIT) is created, which lives on the stack (global to the game loop, essentially). The game loop is as follows:

```
// NOTE(brendan): game loop: event handling -> logic -> rendering
while(gameState.currentState != QUIT) {
	currentTime = SDL_GetTicks();
	elapsedTime = currentTime - previousTime;
	previousTime = currentTime;
	// NOTE(Zach): lag is how much the game's time is behind
	// the real world's time
	lag += elapsedTime;

	// NOTE(Zach): handle events that occur in gameState.currentState
	handleEvents[gameState.currentState](&gameState);

	// NOTE(Zach): loop until the game time is up-to-date with
	// the real time
	while (lag >= MS_PER_UPDATE) {
	// NOTE(Zach): update the game logic of gameState.currentState
		logic[gameState.currentState]();
		lag -= MS_PER_UPDATE;
	}

	// NOTE(Zach): render images that occur in gameState.currentState
	render[gameState.currentState]();
}
```

This loop make three important function calls every frame (frame refreshes are sync'ed to the client's monitor refresh rate):

```
handleEvents[gameState.currentState](&gameState),
logic[gameState.currentState]()
and render[gameState.currentState]()
```

The functions executed by these calls correspond to the game state. The event handling functions are called in this module to handle events, such as mouse clicks. The logic functions are called in the gameLogic module and handle physics updates, such as moving falling tokens along the screen. The rendering functions do the rendering and presenting of textures onto the screen based on the current game state (stored in gameState). Physics updates are done at an interval independent of the frame update by keeping track of the elapsed time since the last physics update, and granularly updating the physics by calling logic[gameState.currentState](); once for N MS_PER_UPDATE intervals where N*MS_PER_UPDATE <= lag < (N+1)*MS_PER_UPDATE.

After the game loop exits (the gameState.currentState became equal to QUIT), the memory allocated for the board is freed using board_destroy(gameState.board), and the memory allocated for the graphics is freed using close_sdl(). 0 is returned for success after the close_sdl() call.

Module: gameLogic

Responsible for implementing the game logic including processing token positioning and game status.

Interface

Uses:

board.h
graphics.h
linkedList.h
SDL.h

Defined Macros/Constants:

NUMBER_OF_STATES: integer
Number of columns on the Connect 4 board.

Global Declarations:

None

Defined Types:

Player: An enumeration with elements: PLAYERONE, PLAYERTWO
Game modes representing a one- or two-player game.

MenuState: An enumeration with the elements: MAINMENU, ONEPLAYER, TWOPLAYER, SETUP, CREDITS, QUIT, DONOTHING.
Game menu options.

GameState: A struct type representing the currentState, currentToken, currentPlayer and the board.

GraphicsState: represents the graphic state.

Access Programs:

setupLogic:
Return Type: void
Parameters: None
Handles droping/falling/positioning of tokens during the game.

transitionSetupTwoPlayer:
Return Type: bool
Parameters: GameState *
Return TRUE if game is IN_PROGRESS, else return FALSE. In addition, print indicate whether the game status is a DRAW, an INVALID_BOARD, a RED_WON or a

BLUE_WON. If the game status is either a RED_WON or a BLUE_WON, highlight the winning tokens.

square:
Return Type: int
Parameters: int x
Return the square of x.

Implementation:

Access Programs:

setupLogic:
Return Type: void
Parameters: None

```
List<FallingToken>::traverseList(clearFallingToken, gFallingTokens);
List<FallingToken>::traverseList(updateFallingToken, 0.5, gFallingTokens);
List<FallingToken>::traverseList(drawFallingToken, gFallingTokens);
List<FallingToken>::traverseList(deleteStillToken, gFallingTokens);
```

transitionSetupTwoPlayer:
Return Type: bool
Parameters: GameState *

```
bool didRedWin = didColourWin(gameState->board, RED);
bool didBlueWin = didColourWin(gameState->board, BLUE);
bool isDraw = checkDraw(gameState->board);
bool isBoardInvalid = checkInvalidBoard(gameState->board);
if(isDraw) {
        printf("Error! The game is a draw!\n");
}
if(isBoardInvalid) {
        printf("Error! Invalid board setup (red tokens - blue tokens > 1)\n");
}
if(didRedWin || didBlueWin) {
        setHighlightedTokenList(getSequentialTokens(gameState->board));
}
if(didRedWin) {
        printf("Error! Red has already won.\n");
}
if(didBlueWin) {
        printf("Error! Blue has already won.\n");
}
if(!(didRedWin || didBlueWin || isDraw || isBoardInvalid)) {
        return true;
}
// NOTE(brendan): game not in progress: continue setup
```

return false;

square:
Return Type: int
Parameters: int x

return x * x

Local Programs:

countTokens:
Return Type: int
Parameters: Board board, Token colour

```
int count = 0
for int row=0; row<NUM_ROWS; row++
        for int col=0; col<NUM_COLS; col++
                if(board_checkCell(board, row, col) == colour)
                        count++
return count
```

didColourWin:
Return Type: bool
Parameters: Board board, Token colour

```
for row=0; row<NUM_ROWS; row++
        for col=0; col<NUM_COLS; col++
                if (board_checkCell(board, row, col) == colour) {
                        //check for same colour in a row horizontally
                        for (currentRow=row, currentCol=col; (currentCol >= 0) &&
                                (board_checkCell(board, currentRow, currentCol) ==
colour);
                                --currentCol) {
                                        if(col – currentCol == 3)
                                                return TRUE
                        }

                        //check for same colour in a row vertically
                        for (currentRow=row, currentCol=col; (currentRow >= 0) &&
                                (board_checkCell(board, currentRow, currentCol) ==
colour);
                                --currentRow) {
                                        if(row – currentRow == 3)
                                                return TRUE
                        }

                        //check for same colour in a row diagonally, decreasing left
                        for (currentRow=row, currentCol=col; (currentRow >= 0) &&
```

```
                                        (currentCol >= 0) &&
                                        (board_checkCell(board, currentRow, currentCol) ==
colour);
                                        --currentRow, --currentCol) {
                                                if(row – currentRow == 3)
                                                        return TRUE
                                }

                                //check for same colour in a row diagonally, decreasing right
                                for (currentRow=row, currentCol=col; (currentRow >= 0) &&
                                        (currentCol >= 0) &&
                                        (board_checkCell(board, currentRow, currentCol) ==
colour);
                                        --currentRow, ++currentCol) {
                                                if(row – currentRow == 3)
                                                        return TRUE
                                }
                        }

                return FALSE //otherwise, return false
```

checkDraw:
Return Type: bool
Parameters: Board board

```
numberOfRedToken = countTokens(board, RED)
numberOfBlueToken = countTokens(board, BLUE)
if numberOfRedToken + numberOfBlueToken == NUM_ROWS * NUM_COLS
        return TRUE
return FALSE
```

checkInvalidBoard:
Return Type: bool
Parameters: Board board

```
numberOfRedToken = countTokens(board, RED)
numberOfBlueToken = countTokens(board, BLUE)
if square (numberOfRedToken - numberOfBlueToken) > 1
        return TRUE
return FALSE
```

equals:
Return Type: bool
Parameters: TokenLocation *tokenA, TokenLocation *tokenB

```
return (tokenA->row == tokenB->row) &&
        (tokenA->colomn == tokenB->column) &&
        (tokenA->colour == tokenB->colour)
```

addNewTokenLocation:
Return Type: List<TokenLocation> *
Parameters: List<TokenLocation> tokenList, int row, int col, Token colour

```
TokenLocation *newHighlightedToken =
(TokenLocation*)malloc(sizeof(TokenLocation));

newHighlightedToken->row = row;
newHighlightedToken->column = column;
newHighlightedToken->colour = colour;

TokenLocation *matchingToken =
List<TokenLocation>::reduceList(equals, newHighlightedToken, tokenList);
 if(matchingToken == NULL) {
        return List<TokenLocation>::addToList(newHighlightedToken, tokenList);
}
return tokenList;
```

getSequentialTokens:
Return Type: List<TokenLocation> *
Parameters: Board board

```
List<TokenLocation> *sequentialTokens = NULL;
 for(int row = 0; row < NUM_ROWS; ++row) {
        for(int col = 0; col < NUM_COLS; ++col) {
                Token firstColour = board_checkCell(board, row, col);
                 if(firstColour != EMPTY) {
                // NOTE(brendan): Check for 4-in-a-row in a row starting at the
                // (row, col) token
                        for(int currentCol = col, currentRow = row;
                                (currentCol >= 0) &&
                 (board_checkCell(board, currentRow, currentCol)) == firstColour;
                                --currentCol) {
                                        if(col - currentCol == 3) {
                                         while(currentCol <= col) {
                  sequentialTokens = addNewTokenLocation(sequentialTokens, row,
                   currentCol++, firstColour);
          }
         break;
                }
        }

 // NOTE(brendan): Check for 4-in-a-row in a column
for(int currentCol = col, currentRow = row;
         (currentRow >= 0) &&
         (board_checkCell(board, currentRow, currentCol)) == firstColour;
         --currentRow) {
         if(row - currentRow == 3) {
                while(currentRow <= row) {
```

```
                        sequentialTokens = addNewTokenLocation(sequentialTokens,
                    currentRow++, col, firstColour);
                    }
                    break;
                    }
        }

    // NOTE(brendan): check for 4-in-a-row diagonal decreasing left
    for(int currentCol = col, currentRow = row;
        (currentRow >= 0) && (currentCol >= 0) &&
        (board_checkCell(board, currentRow, currentCol)) == firstColour;
        --currentRow, --currentCol) {
      if(row - currentRow == 3) {
        while(currentRow <= row) {
          sequentialTokens = addNewTokenLocation(sequentialTokens,
              currentRow++, currentCol++, firstColour);
        }
        break;
      }
    }

    // NOTE(brendan): check for 4-in-a-row diagonal increasing left
    for(int currentCol = col, currentRow = row;
        (currentRow >= 0) && (currentCol < NUM_COLS) &&
        (board_checkCell(board, currentRow, currentCol)) == firstColour;
        --currentRow, ++currentCol) {
      if(row - currentRow == 3) {
        while(currentRow <= row) {
          sequentialTokens = addNewTokenLocation(sequentialTokens,
              currentRow++, currentCol--, firstColour);
        }
        break;
      }
    }
  }
 }
}

        return sequentialTokens;
```