Crosswinds

Software Design Description

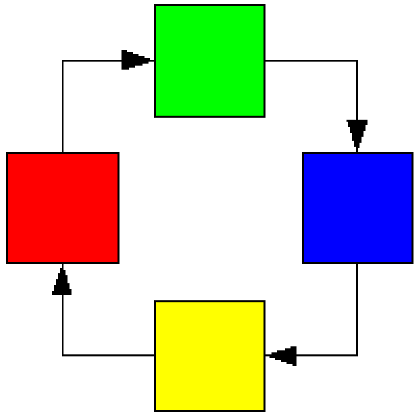
Keith Co (co-lead: Networking module)
Roger Han (lead: Game module)
Dan Harris (co-lead: Rendering module)
Charlene Jeune (co-lead: Rendering module)
Tim Jones (co-lead: Interaction module)
Brandon Kelch (co-lead: Interaction module)
Gary Mgerian (lead: Control module)
Noel Overkamp (co-lead: Networking module)

Alex Francois (Project Manager)

Table of Contents

1. System Overview

The operational game system will consist of 2-4 identical sub-systems connected to each other in a ring topology. The software nodes communicate using an ad hoc protocol.



The architecture for the game system is specified in the SAI architectural style:

A good number of elements are already defined and implemented in the FSF library and in existing modules:

- GlutIO module for OpenGL rendering and input from keyboard and joystick/gamepad
- TcpIp module for network socket communication

All other elements will be specified and implemented for this project.

Commons:
Constants
Some data structures (piece, token)

Modules:

- Events - Protocol for internal communication (event nodes), game events, interaction events, chat events
- Game - game (state) node, game update cell
- Interaction - interaction state node, interaction update cell
- Rendering - render cell (render game state, interaction, chat data)
- Networking - filter messages, encode/decode buffers from/to events
- Control - encode keyboard and gamepad input into interaction events
- Chat - chat data, receive and compose message,

2. Constants and Common Objects

Namespace: crosswinds
File: crosswinds.h
External dependencies: none
Internal Dependencies: none

Constants

Pieces:

```
const int NB_PIECES=20
const int NO_PIECE=-1
```

Coordinate ranges:

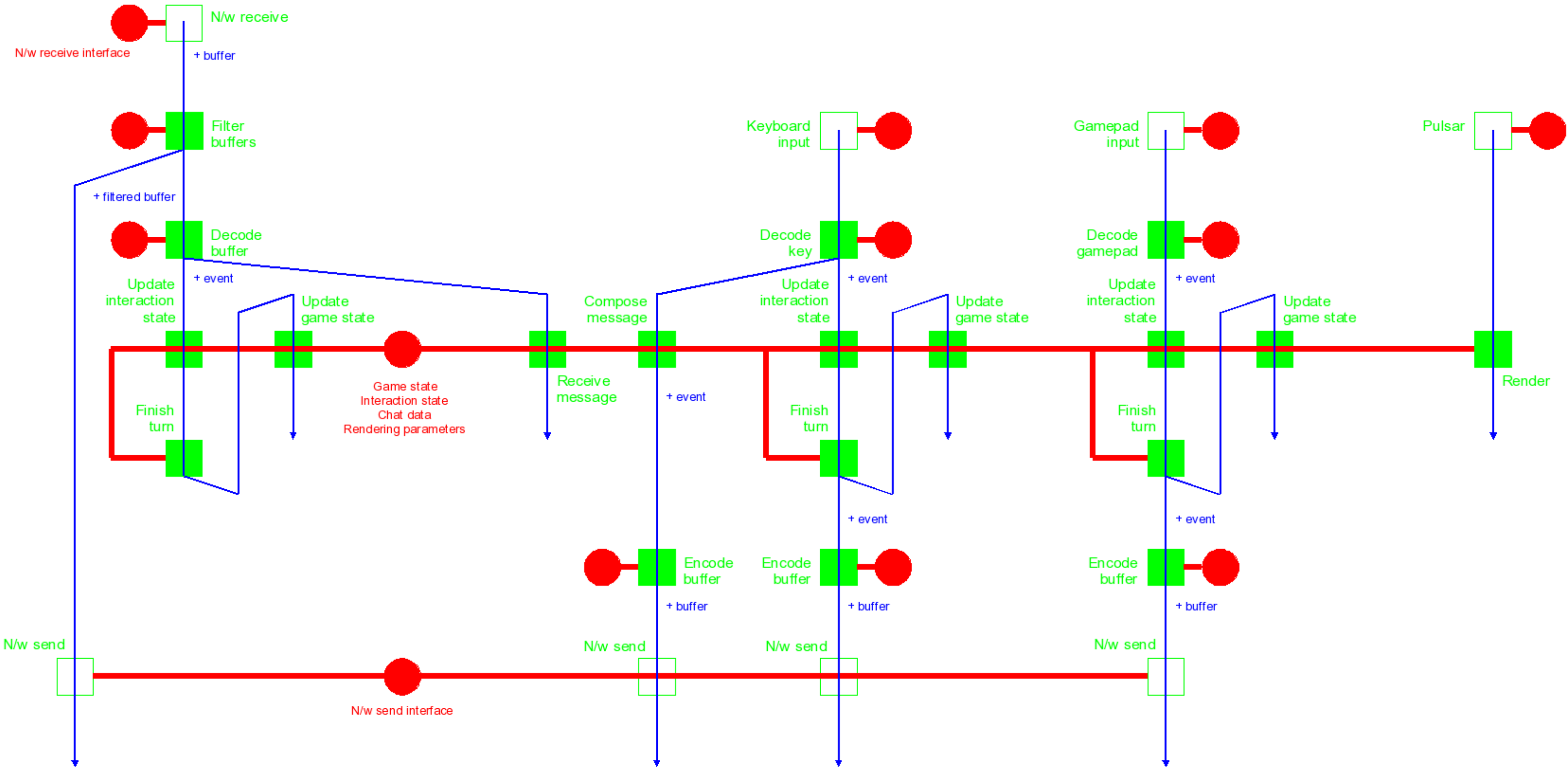
```
const int X_MIN=0
const int X_MAX=21
const int Y_MIN=0
const int Y_MAX=21
```

Coordinates of color squares:

```
const int X[4]={10,10,11,11}
const int Y[4]={10,11,11,10}
```

Colors are used to specify direction and orientation on the board, as well as player identity and piece ownership:

```
typedef int Color;
```



const crosswinds::Color crosswinds::RED=0
const crosswinds::Color crosswinds::GREEN=1
const crosswinds::Color crosswinds::BLUE=2
const crosswinds::Color crosswinds::YELLOW=3

Rotation relative directions:
const int crosswinds::LEFT=0
const int crosswinds::RIGHT=1
const int crosswinds::UP=2
const int crosswinds::DOWN=3

Piece Types:
const int crosswinds::PIECE_J=0; A J-piece is identified when the arrow is pointing up and the other end is pointing to the left.
const int crosswinds::PIECE_L=1; An L-piece is identified when the arrow is pointing up and the other end is pointing to the right.
const int crosswinds::PIECE_T=2;
const int crosswinds::PIECE_I=3;

const int crosswinds::UNDEFINED=-1

```
Game Update: //Keith's Stuff
    const int CLOCKWISE = 1;
    const int COUNTERCLOCKWISE = -1;
    const int DEFAULT_SPEED = 4;
    const int POINTS_TO_WIN = 3;
    const int PIECE_XPOSITION[20] = {-2, -2, -2,  -2, -3,  4,  8, 12, 14, 19, 23, 23, 23, 23, 24, 17, 13, 9,  7,   2};
    const int PIECE_YPOSITION[20] = { 4,  8,  12, 14, 19, 23, 23, 23, 23, 24, 17, 13,  9,  7,  2, -2, -2, -2, -2, -3};
    const int PIECE_TYPE[5] = {PIECE_T, PIECE_T, PIECE_L, PIECE_J, PIECE_I};
    const int PIECE_ORIENTATION[20] = {
                          (RED + CLOCKWISE*2)%4,
                          (RED + CLOCKWISE*2)%4,
                          (RED + CLOCKWISE*2)%4,
                          (RED + CLOCKWISE*2)%4,
                          (RED + CLOCKWISE)%4,

                          (GREEN + CLOCKWISE*2)%4,
                          (GREEN + CLOCKWISE*2)%4,
                          (GREEN + CLOCKWISE*2)%4,
                          (GREEN + CLOCKWISE*2)%4,
                          (GREEN + CLOCKWISE)%4,

                          (BLUE + CLOCKWISE*2)%4,
                          (BLUE + CLOCKWISE*2)%4,
                          (BLUE + CLOCKWISE*2)%4,
                          (BLUE + CLOCKWISE*2)%4,
                          (BLUE + CLOCKWISE)%4,

                          (YELLOW + CLOCKWISE*2)%4,
                          (YELLOW + CLOCKWISE*2)%4,
                          (YELLOW + CLOCKWISE*2)%4,
                          (YELLOW + CLOCKWISE*2)%4,
                          (YELLOW + CLOCKWISE)%4
                          };
```

Data structures

crosswinds::CPiece
int m_x : x coordinate of position
int m_y : y coordinate of position
int m_type : Type of piece
crosswind::Color m_orientation : piece orientation
crosswind::Color m_owner : piece owner
bool m_inPlay : flag, true if piece is in play (on the board)

crosswinds::CToken
int m_x : x coordinate of position [X_MIN,X_MAX]
int m_y : y coordinate of position [Y_MIN,YMAX]
crosswinds::Color m_m : token motion direction

3. EventsModule

Namespace: events
Files: EventsModule.[h|cpp]
External dependencies: none
Internal Dependencies: Commons

Constants

events::CEventBase (fsf::CNode) : EVENTS_BASE
char m_id : event ID
base class for all events, implements protocol overhead

Event IDs:

const char EVENTS_BASE=0

const char EVENTS_GAME=10

```
const char EVENTS_GAME_COMMIT=11
const char EVENTS_GAME_PLAYER_QUIT=12
const char EVENTS_GAME_WIN_ROUND=13
const char EVENTS_GAME_WIN_MATCH=14
(removed EVENTS_GAME_PASS)

const char EVENTS_INTERACTION=20
const char EVENTS_INTERACTION_SELECT=21  // ignore for now
const char EVENTS_INTERACTION_ UNSELECT=22
const char EVENTS_INTERACTION_SELECT_NEXT=23
const char EVENTS_INTERACTION_SELECT_PREVIOUS=24
const char EVENTS_INTERACTION_TRANSLATE=25
const char EVENTS_INTERACTION_ROTATE=26
const char EVENTS_INTERACTION_REMOVE=27
const char EVENTS_INTERACTION_PASS=28
const char EVENTS_INTERACTION_COMMIT=29

const char EVENTS_CHAT=50
const char EVENTS_CHAT_MESSAGE=51
const char EVENTS_CHAT_CHARACTER=52
```

Game event nodes

events::CGameEventBase (events::CEventBase) : EVENTS_GAME
base class for game events

events::CGameCommit (events::CGameEventBase) : EVENTS_GAME_COMMIT
int m_piece : index of manipulated piece [0,NB_PIECES-1] - NO_PIECE if none
int m_piece_x : x coordinte of new piece position [X_MIN+1,X_MAX-1] - undefined if no manipulated piece specified
int m_piece_y : y coordinate of new piece position [Y_MIN+1,Y_MAX-1] - undefined if no manipulated piece specified
crosswinds::Color m_piece_o : piece orientation - undefined if no manipulated piece specified
int m_token_x : x coordinate of new token position [X_MIN,X_MAX]
int m_token_y : y coordinate of new token position [Y_MIN_Y_MAX]
crosswinds::Color m_token_m : token direction
int m_removedSelected : index of removed piece by player choice [0,NB_PIECES-1] - NO_PIECE if none
int m_removedForced : index of removed piece due to being hit twice [0,NB_PIECES-1] - NO_PIECE if none

(removed:
events::CGamePass (events::CGameEventBase) : EVENTS_GAME_PASS)

events::CGamePlayerQuit (events::CGameEventBase) : EVENTS_GAME_PLAYER_QUIT
crosswinds::Color m_player : player id

events::CGameWinRound (events::CGameEventBase) :EVENTS_GAME_WIN_ROUND
crosswinds::Color m_player : player id

events::CGameWinMatch (events::CGameEventBase) : EVENTS_GAME_WIN_MATCH
crosswinds::Color m_player : player id

Interaction event nodes

events::CInteractionEventBase (events::CEventBase) : EVENTS_INTERACTION
base class for interaction events

events::CInteractionSelect (events::CInteractionEventBase) : EVENTS_INTERACTION_SELECT
int m_piece : index of piece (0-19)

events::CInteractionUnselect (events::CInteractionEventBase) : EVENTS_INTERACTION_ UNSELECT

events::CInteractionSelectNext (events::CInteractionEventBase) :
EVENTS_INTERACTION_SELECT_NEXT

events::CInteractionSelectPrevious (events::CInteractionEventBase) :
EVENTS_INTERACTION_SELECT_PREVIOUS

events::CInteractionTranslate (events::CInteractionEventBase) : EVENTS_INTERACTION_TRANSLATE
crosswinds::Color m_direction : absolute direction (RED, GREEN, BLUE or YELLOW)

events::CInteractionRotate (events::CInteractionEventBase) : EVENTS_INTERACTION_ROTATE
int m_direction : relative direction (CLOCKWISE or COUNTERCLOCKWISE)

events::CInteractionRemove (events::CInteractionEventBase) : EVENTS_INTERACTION_REMOVE

events::CInteractionPass (events::CInteractionEventBase) : EVENTS_INTERACTION_PASS

events::CInteractionCommit (events::CInteractionEventBase) : EVENTS_INTERACTION_COMMIT

Chat event nodes

events::CChatEventBase (events::CEventBase) : EVENTS_CHAT
base class for chat events

events::CChatMessage (events::CChatEventBase) : EVENTS_CHAT_MESSAGE

events::CChatCharacter (events::CChatEventBase) : EVENTS_CHAT_CHARACTER

4. GameModule

Namespace: game
Files: GameModule.[h|cpp]
External dependencies: none
Internal Dependencies: Commons

Game node

game::CState (fsf::CNode) : GAME_STATE

Data members:
fsf::CMutex m_csState : mutex for state object
int m_nbPlayers
int m_scores[4];
crosswinds::Color m_activePlayer : currently active player
int m_tokenSpeed : number of steps token will move in this turn (1-20)
crosswinds::CPiece m_pieces[20] : array of pieces
crosswinds::CToken::m_token : token
m_board[22][22] : Collision detection board. Rows & Columns 0 & 21 (Outer edges) are goals.
A location with the token on it has a value of 100 or more.

A location with a piece has value of the index of that piece.
The four middle locations have a value of 50.
An empty space has a value of -1.
bool m_matchWinState: flag true if someone has won the game and a new game has not yet started
int m_matchWinCounter: counter for duration of match win state
crosswinds::Color m_matchWinner: most recent winner of match

Member functions:

void IncrementTokenSpeed() - Increments token speed by one.
void ResetRound() - Resets token position and speed as well as active player.
void ChangePlayer() - Makes the player to the right the active player.
void IncrementPlayerScore(crosswinds::Color) - Increments a player's score by one.
void Init() - Resets all pieces to game start positions.
int RandomNumber(int maxNum) - Generates a random number from 0 to maxNum.
void SetBoardPiece(int x, int y, int o, bool inPlay, int type, int piece) - X is xposition of piece, y is yposition of piece, o is orientation, inPlay is whether the piece is in play or not, type is the type of piece, piece is the index of the piece. This sets the collision board.
void SetBoardToken(int x, int y, bool moved) - This sets the token on the collision board. x is the xposition, y is the yposition, moved is the variable that decides whether to remove or add the token to the board at the current position.
void matchWin(bool w) - Sets whether the game is in the match win state and sets a counter for length of the state's duration
int matchWinCounter() - Decrements the counter, when counter reaches zero match win state is set back to false
void matchWinner(crosswinds::Color winner) - Sets who is the most recent match winner
void ResetScores() - Resets the score of all players to 0.

Game update cell

game::CUpdate (fsf::CCell)	GAME_UPDATE
Active filter	[FSF_ACTIVE_PULSE "Root" [GAME_EVENT_BASE "Event" *EVENT*] [GAME_WIN_ROUND "Win round" *WIN ROUND*]* [GAME_WIN_MATCH "Win match" *WIN MATCH*]*]
Passive filter	[GAME_STATE "Game state"]
Output	()

Finish turn cell

game::CFinishTurn (fsf::CCell)	GAME_FINISH_TURN
Active filter	[FSF_ACTIVE_PULSE "Root"]
Passive filter	[GAME_STATE "Game state"]
Output	()

5. InteractionModule

Namespace: interaction
Files: InteractionModule.[h|cpp]
External dependencies: none
Internal Dependencies: Commons, GameModule

Interaction node

interaction::CState (fsf::CNode) : INTERACTION_STATE

Data members:
fsf::CMutex m_csState : mutex for state object
bool m_active : flag true if player is interacting (otherwise only watching another player play their turn)
crosswinds::CToken m_token : predicted (position and motion direction of) token
int m_selectedPiece : index of currently selected piece [0,NB_PIECES-1], NO_PIECE if none
crosswinds::CPiece : currently manipulated piece (values undefined if no currently selected piece)
int m_removedPiece : index of removed piece [0,NB_PIECES-1], NO_PIECE if none
bool m_validState : flag true if current state is valid
bool m_errorState: flag true if error generated
int m_errorCounter: counter for duration of error state

Additional array for path planning:
int m_direction[22][22]

Member functions:
bool Active(): return m_active
const crosswinds::CToken& Token(): return m_token
int SelectedPiece(): return m_selectedPiece
crosswinds::CPiece& Piece(): return m_piece
int RemovedSelectedPiece(): return m_removedSelected
int RemovedForcedPiece(): return m_removedForced
bool ValidState(): return m_validState
crosswinds::Color MyColor(): return m_myColor
bool ErrorState(): return m_errorState

void Valid() : set the state to valid
void InvalidMove(): set the state to invalid
void RemovePiece(): set the removedPiece index to the selectedPiece index
void CancelRemovePiece(): set the removedPiece index to NO_PIECE
void UnselectPiece(): set the selectedPiece index to NO_PIECE
void IncrementSelectedPiece(): increment selectedPiece
void DecrementSelectedPiece(): decrement selectedPiece

Translation/Rotation

Currently selected piece's new position is calculated. The new position is checked against the board to see if its a valid position and the "valid" variable is set accordingly.   A player can only commit if the valid variable is true.

Select Next/Previous

Increment/Decrement the SelectedPiece index only including pieces that are either in play or belong to the owner.

Token path planning

The direction array encodes at each position the direction imposed by a piece (color) or lack thereof (crosswinds::NO_PIECE). The coordinate system is the same as for rendering.

The token prediction data is maintained as follows:

- Initially, there are no pieces on the board
- The data is updated according to the interaction event
    - Select Next/Previous
        - If the selected piece is not in play: ClearSelectedDirections; select next piece; WriteSelectedDirections; ComputeToken
        - If the selected piece is in play: (WriteSelectedDirections;) select next piece; WriteSelectedDirections; ComputeToken
    - Unselect:
        - If the selected piece is not in play: ClearSelectedDirections; unselect piece; ComputeToken
        - If the selected piece is in play: unselect piece
    - Translate or Rotate: ClearSelectedDirections; move/rotate piece; WriteSelectedDirections; ComputeToken
    - Remove: ClearSelectedDirections; ComputeToken
    - Commit: ComputeToken (in case it was not computed before)

where:

- ClearSelectedDirections: if the previous state was valid and a piece is currently selected, reset direction under currently selected piece
- WriteSelectedDirections: if the current state is valid and a piece is currenlty selected, set direction under currently selected piece and set previous state to valid
- ComputeToken: compute the new token position

The path planning algorithm is as follows:

- Start from current token position
- repeat
    - move token in direction stored at current token location in direction array
    - until token is in a goal area or number of moves (token speed) reached

Interaction update cell

interaction::CUpdate (fsf::CCell)	INTERACTION_UPDATE
Active filter	[EVENTS_INTERACTION "*"]
Passive filter	[FSF_PASSIVE_PULSE "Root" [GAME_STATE "Game state"] [INTERACTIOIN_STATE "Interaction state"]]
Output	()

6. RenderingModule

Namespace: rendering
Files: RenderingModule.[h|cpp]
External dependencies: GlutIOModule
Internal Dependencies: GameModule, InteractionModule, ChatModule

Rendering cell

rendering::CRender (fsf::CCell)	RENDERING_RENDER
Active filter	[FSF_ACTIVE_PULSE "Root"]
Passive filter	[FSF_PASSIVE_PULSE "Root" [RENDERING_CAMERA "Camera"] [GAME_STATE "Game state"] [INTERACTION_STATE "Interaction state"]]
Output	()

Member Functions:

void drawBoard() - Draws the game board including player goals

void draw_starting_points() - draws the four starting positions

void draw_path() - Draws a path starting at the token.

void draw_square(double x, double y, double z) - A simple function to draw the squares of the pieces

void draw_arrow(double pieceX, double pieceY, double pieceZ, crosswinds::Color dir) - Draws an arrow pointing in the appropiate direction for a piece

void draw_pieces(game::CState *pGS, interaction::CState *pIS) - Draws all the pieces according to the game state and interaction state. Calls the four different piece drawing functions.

void draw_tPiece(double x, double y, double z, crosswinds::Color dir,bool selected,bool shadow) - Draws T-shaped piece

void draw_iPiece(double x, double y, double z, crosswinds::Color dir,bool selected,bool shadow) - Draws I-shaped piece

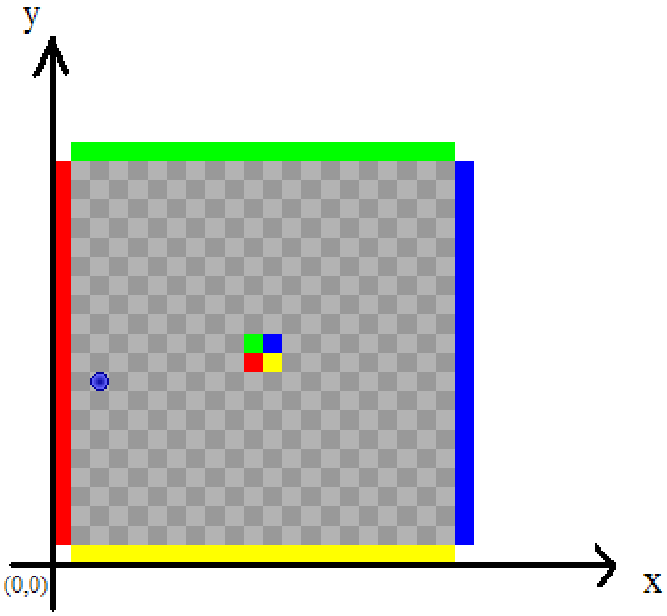
void draw_jPiece(double x, double y, double z, crosswinds::Color dir,bool selected,bool shadow) - Draws J-shaped piece

void draw_lPiece(double x, double y, double z, crosswinds::Color dir,bool selected,bool shadow) - Draws L-shaped piece

void draw_circle(double x, double y, double z, double r) - Draws a circle

void draw_score(double x, double y, double r) - Draws a single score indicator circle

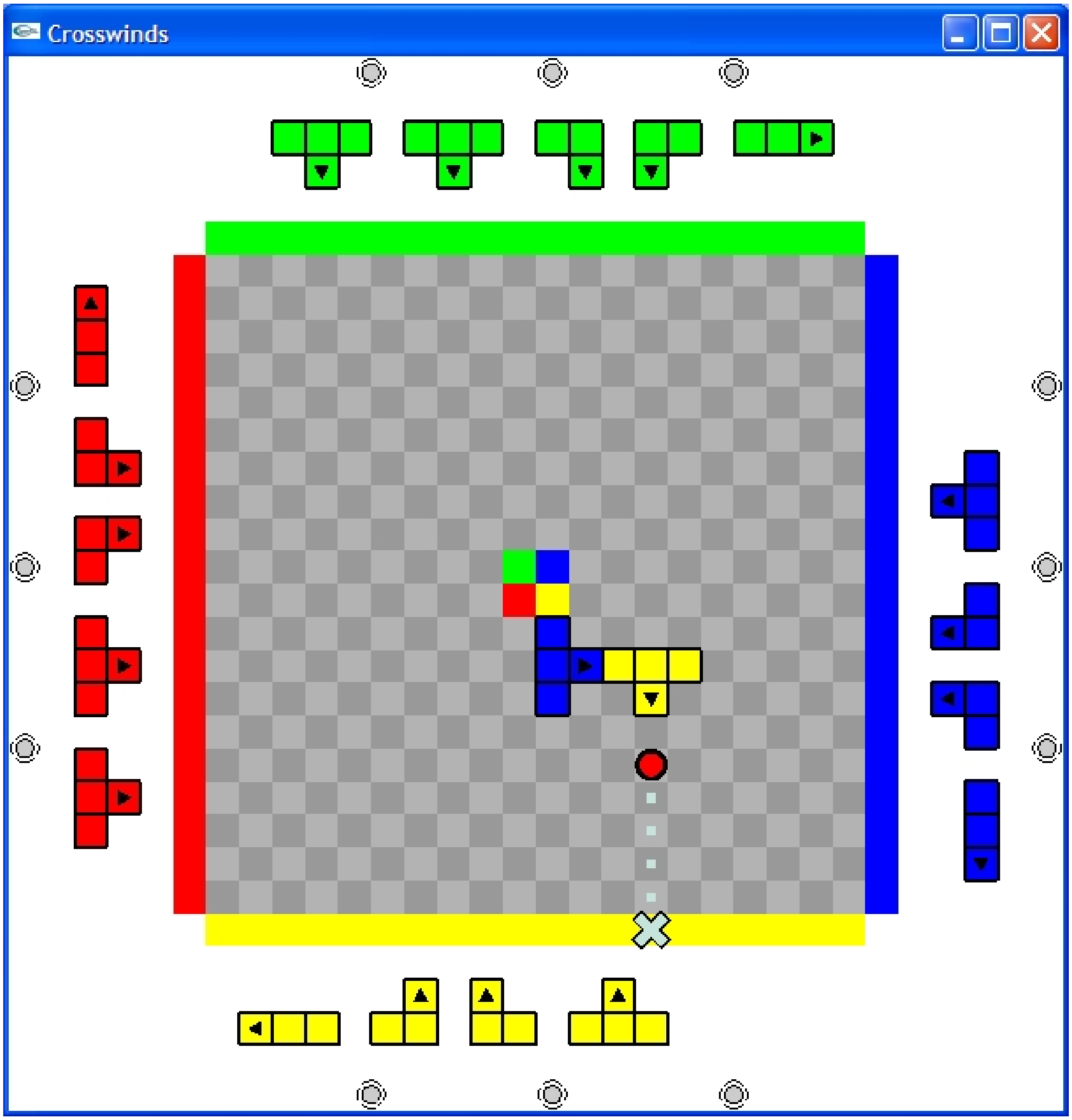
void draw_scores(game::CState *pGS, interaction::CState *pIS) - Draws scores for all players



The main playing area is rendered from (1,1) to (21,21)
The red goal is from (0,1) to (1,21)
The yellow goal is from (1,0) to (21,1)
The blue goal is from (21,1) to (22,21)
The green goal is from (1,21) to (21,22)
Squares are 1x1.

The current camera view is orthographic.

When a piece is moved, a shadow piece appears in its previous place.



7. NetworkingModule

Namespace: networking
Files: NetworkingModule.[h|cpp]
External dependencies: GlutIOModule
Internal Dependencies: Commons

Constants

```
// Amount of data to store for each type of event

// All data is stored as char



const int EVENTS_INTERACTION_TRANSLATE_SIZE = 3*sizeof(char);
const int EVENTS_INTERACTION_ROTATE_SIZE = 3*sizeof(char);

const int EVENTS_INTERACTION_REMOVE_SIZE = 2*sizeof(char);

const int EVENTS_INTERACTION_SELECT_NEXT_SIZE = 2*sizeof(char);

const int EVENTS_INTERACTION_SELECT_PREVIOUS_SIZE = 2*sizeof(char);

const int EVENTS_INTERACTION_SELECT_SIZE = 3*sizeof(char);

const int EVENTS_INTERACTION_UNSELECT_SIZE = 2*sizeof(char);

const int EVENTS_INTERACTION_PASS_SIZE = 2*sizeof(char);

const int EVENTS_INTERACTION_COMMIT_SIZE = 2*sizeof(char);

const int EVENTS_CHAT_CHARACTER_SIZE = 2*sizeof(char);

const int EVENTS_CHAT_MESSAGE_SIZE = 2*sizeof(char);



// Protocol for encoding from a events class to a CCharBuffer (char *pData):


   pData[0]=me   (player which the event originated from)

   pData[1]=static_cast<char>(EVENTS_TYPE i.e. EVENTS_INTERACTION_ROTATE)

   pData[2]=static_cast<char>(data member #1 of EVENTS_CLASS)

   ...

   pData[n]=static_cast<char>(data member #n of EVENTS_CLASS)



// Data should be in the order of the constructor of the events class

EVENTS_INTERACTION_SELECT

  pData[0] -- me

  pData[1] -- EVENTS_INTERACTION_SELECT


EVENTS_INTERACTION_UNSELECT

  pData[0] -- me
```

pData[1] -- EVENTS_INTERACTION_UNSELECT

EVENTS_INTERACTION_SELECT_NEXT

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_SELECT_NEXT

EVENTS_INTERACTION_SELECT_PREVIOUS

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_SELECT_PREVIOUS

EVENTS_INTERACTION_TRANSLATE

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_TRANSLATE

pData[2] -- m_direction

EVENTS_INTERACTION_ROTATE

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_ROTATE

pData[2] -- m_direction

EVENTS_INTERACTION_REMOVE

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_REMOVE

EVENTS_INTERACTION_PASS

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_PASS

EVENTS_INTERACTION_COMMIT

pData[0] -- me

pData[1] -- EVENTS_INTERACTION_COMMIT

```
// Protocol for decoding from a CCharBuffer to an event class

     char *pBufferData  // read from CCharBuffer

   if(static_cast<char>(pBufferData[0])==EVENTS_TYPE){

       // Create the appropriate event object feeding data into constructor.  This is done in the order
shown above

       //The last item in the constructor will be pActivePulse

   }

//eg.
   if(static_cast<char>(pBufferData[0])==EVENTS_GAME_WIN_ROUND){
       CGameWinRound(pBufferData[1],
                           pActivePulse);
   }
```

Filter buffer cell

networking::CFilterBuffer (fsf::CCell)	NETWORKING_FILTER_BUFFER
Active filter	[FSF_CHAR_BUFFER "buffer"]
Passive filter	[FSF_INT32_NODE "Me"]
Output	(FSF_CHAR_BUFFER "Transmit buffer")

This cell takes as input a character buffer (received on the network) and creates a copy with a different name (default "Transmit buffer") if the message originated from a different player. The goal is to identify those messages that require additional processing (new name) from those that originated from the local system (and thus have gone around the ring already). This cell only needs to process one buffer at a time.

Decode buffer cell

networking::CDecode	NETWORKING_DECODE

Active filter	[FSF_CHAR_BUFFER "Buffer"]
Passive filter	[FSF_PASSIVE_PULSE "Root"]
Output	(EVENTS_BASE "Event")

Encode buffer cell

networking::CEncode	NETWORKING_ENCODE
Active filter	[FSF_ACTIVE_PULSE "Root"]
Passive filter	[FSF_INT32 "Me"]
Output	(FSF_CHAR_BUFFER "buffer")

8. ControlModule

Namespace: control
Files: ControlModule.[h|cpp]
External dependencies: GlutIOModule
Internal Dependencies: EventsModule, InteractionModule

Decode keyboard cell

control::DecodeKeyboard (fsf::CCell)	CONTROL_DECODE_KEYBOARD
Active filter	[GLUTIO_KEYBOARD_EVENT "Keyboard event"]
Passive filter	[FSF_INT32_NODE "Me"]
Output	(EVENTS_INTERACTION "Event")

The Active filter for the keyboard, called the "Keyboard Event" is the GlutIO Keyboard Event. The keyboard control uses the FSF_INT32_Node title "Me", and everytime the attempts to make an input by pressing a button, and output is created which is the Interaction Event title "Event.

Keyboard Input/Game Controller (Gary Mgerian)
The user will interact with the game through the keyboard. All of the desired moves and selections are made by inputting designated keys which perform commands such as commit move, select piece, quit game,etc.

The Commands are as follows:

COMMAND	***KEY(S)***
Commit	c, C "Enter"
Unselect	u, U, t, s
Select Next Piece (iterate through the pieces)	f, >, n, N
Select Previous Piece (iterate though the pieces)	d, p, P, <, b
Translate	arrow keys (up,down,left,right)
Rotate clockwise	r

Rotate counterclockwise	e
Remove a piece	R
Quit	q

gamepad cell

control::DecodeGamepad (fsf::CCell)	CONTROL_DECODE_GAMEPAD
Active filter	[GLUTIO_JOYSTICK_EVENT "Joystick event"]
Passive filter	[FSF_INT32_NODE "Me"]
Output	(EVENTS_INTERACTION "Event")

The Active filter for the gamepad is called "Joystick event" and it is a GlUTIO Joystick Event. The game controller uses the FSF_INT32_NODE title "ME". Whenever the user enters a button, and output is made which is a Interaction Event title "Event".

The Commands for the Game Controller are as follows:

<table>
<tr><th><em>COMMAND</em></th><th><em>KEY(S)</em></th></tr>
<tr><td>Commit</td><td>'O'</td></tr>
<tr><td>Unselect</td><td>'^̲' (triangle button)</td></tr>
<tr><td>Select Next Piece (iterate through the pieces)</td><td>'R2'</td></tr>
<tr><td>Select Previous Piece (iterate though the pieces)</td><td>'R1'</td></tr>
<tr><td>Translate</td><td>arrow keys (up,down,left,right)</td></tr>
<tr><td>Rotate clockwise</td><td>'X'</td></tr>
<tr><td>Remove a piece</td><td>__
|_| ("square button")</td></tr>
</table>

9. ChatModule

Namespace: chat
Files: ChatModule.[h|cpp]
External dependencies: none
Internal Dependencies: EventsModule, InteractionModule

Chat data

chat::CMessageBuffer (fsf::CCharBuffer) : CHAT_MESSAGE_BUFFER

chat::CComposeBuffer (fsf::CCharBuffer) : CHAT_COMPOSE_BUFFER

Receive message cell

chat::Receive (fsf::CCell)	CHAT_RECEIVE
Active filter	[EVENTS_CHAT_MESSAGE "Message"]

Passive filter	[CHAT_MESSAGE_BUFFER "Message buffer"]
Output	()

Compose message cell

chat::Compose (fsf::CCell)	CHAT_COMPOSE
Active filter	[EVENTS_CHAT_CHARACTER "Character"]
Passive filter	[CHAT_COMPOSE_BUFFER "Compose buffer"]
Output	(EVENTS_CHAT_MESSAGE "Message"]

10. Development Plan

Scope of the prototype

The goal is to develop a fully functional prototype that implements the game interface and mechanics for four networked players engaged in a single match. See Appendix B for additional notes and restrictions.

Schedule

Preparation	**Week 1**	**Week 2**	**Week 3 (*)**	**Week 4**
Events				
Data structures (Game, Interaction)				
	Rendering (*) - C,D	Rendering - C,D	Rendering - C,D	
	Game update - K	Game update (*) - K	Finish turn - K	
	Control (*) - G			
	Interaction update - B,T	Interaction update (*) - B,T,G		
	Networking - N,R	Networking - N,R	Networking (*) - N,R	
	(Chat)	(Chat)	(Chat)	

(*) Critical milestone

Appendix A - Module Dependencies

Comons
Events: Commons
Game: Commons, Events
Interaction: Commons, Events, Game
Control: Commons, Events
Networking: Commons, Events

Chat: Commons, Events
Rendering: Commons, Game, Interaction, Chat

Appendix B - Additional Development Notes

TCPIP Connections

Actual network connectivity is provided by an existing TCPIP MFSM module, which makes use of Windows Sockets. In order to build a ring of four instances of the player system, each instance receives data on a predefined port that is a function of the player ID (the numerical value associated with the player's color). The current prototype listens on port 2000+color. The prototype attempts to connect to the next element on the ring given its IP address (the port is computed automatically).

Actual discovery and join in are out of scope; connections are established manually. This aspect of the system would require additions to the message protocol to handle related events.

Initialization

Match initialization requires a special protocol to pass along and synchronize initial state (which involves random elements). This protocol would relate to that invoved in creating the ring (see last paragraph). This is out of scope of the present prototype.