Christopher Hernandez – Lead Design
Erin Barker – Content Design
Ting Kong – Level Design
Harjot Singh – System Design
Luis Jimenez – User Interface Design
**REQUESTS**

**DESCRIPTION & OBJECTIVE**

Scrambled Net is an enjoyable puzzle game that can become challenging as the user progresses through stages. The objective of the game is to turn on and connect all the displayed terminals to the given server(s). The game begins by displaying a welcome screen that shows the user a list of options to choose from. Options such as “Play”, “Exit”, and “View Scores” can be selected by the user from the menu screen. If the user selects “Play”, he or she will then be given a grid of scrambled pipes, terminals, and server(s) based on the selected difficulty. In order for the player to move on to the next stage he or she has to create a connection using the pipes. In order to create a proper connection the pipes, terminals, and server(s) can be rotated clockwise by clicking on them. After the user has completed the first stage he or she can move on to the next stage or view the scoreboard by selecting that option. The scoreboard will contain other player’s score of the stages that have been reached. If the user decides to quit at anytime the option of quitting is always available throughout the game.

**KEY TERMS**

Throughout the report key phrases will be used to refer to the different objects that will be placed in the cells of the grid.

![Image of L Pipe, T Pipe, X Pipe, Dash Pipe, Monitor, and Server](image)

**INPUT**

During the course of game the computer needs to be able to receive input from the user to properly function and allow the user to enjoy the game. Two main sources of input that are used throughout the game are the keyboard and the mouse.

**Mouse Input**

During game play the user is allowed to click on any cell in the grid to rotate the object into its desired position. This allows the user to create different paths to connect the monitor to the server. Figure 1 shows an example of what an L Pipe will look like when clicked on.

![Click #1 Click #2 Click #3 Click #4](image)

*Fig. 1*

**Keyboard Input**

In order for the user to be able to select an option that is listed in the main menu, the user needs to be able to enter a key to make a selection. This feature will allow the user to navigate through the game. Figure 2 has shows a diagram of the different paths created from the options throughout the game.
**ALGORITHM**

During game play an algorithm needs to properly detect a connection with all terminals to the server(s). The algorithm makes sure that all terminals are connected without any terminals left without a connection. Figure 3 gives an example of a full connection. If there is any terminal remaining without a connection then the structure of the pipes, terminals, or server(s) is incorrect.

**FULLFILED REQUIREMENTS**

Fulfilling the needed requirements creates an enjoyable game for the user. After all requirements have been properly fulfilled, the designing process of the game can begin. By following the requirements the designers of the game have a small set of rules to follow and help design the project.
Level Design

Level 1 – Small Grid

Level 2 – Medium Grid
Teamwork

1. Lead Design - Christopher Hernandez
   Lead design is the general management and oversight of the project. For this project, lead design is the main contact and organizer of meet ups.

2. Content Design – Erin Barker
   Content design is the creation of characters, items, puzzles, and missions. For this game, there are puzzle pieces of various shaped pipes, monitors, and servers.

3. Level Design – Ting Kong
   Level design is the construction of world levels and its features. This game has three levels. Each level has a different amount of puzzle pieces. The levels have varying layouts as well.

4. System Design – Harjot Singh
   System design is the creation of game rules and underlying mathematical patterns. This is where the complex algorithm that detects a winning route, and the AI creation for solving a level is done.

5. User Interface Design – Luis Jimenez
   User Interface design constructs the user interactions and feedback interface, like menus or heads-up displays. This game will have an opening menu, and displays when a user has won the puzzle.

Our team is effective because we communicate, participate, and are fully engaged. When we schedule a group meeting, we make sure that every member agrees to the day and time of the meeting. If somebody is unable to meet at the last minute, we make sure they still know what is going on with the group. Every member of our team is eager to participate in the projects. We all put in what we are able to.

There are ten qualities of an effective team member; Demonstrates reliability, Communicates constructively, Listens actively, Functions as an active participant, Shares openly and willingly, Cooperates and pitches in to help, Exhibits flexibility, Shows commitment to the team, Works as a problem-solver, Treats others in a respectful and supportive manner. Our team members each have these qualities, but the best quality that each of us has is:

- Chris is good at communicating constructively. If he feels that there is an issue with something, he can find a way to explain what is going on without stepping on anybody’s toes.
- Ting always cooperates and pitches in to help. She has experience that is useful to the team and likes to share her awesome creations.
- Jay (Harjot) is best at demonstrating reliability. He will get the task at hand taken care of. The other team members don’t have to worry if he’ll get something done, because he always does.
- Luis exhibits flexibility. He has an overwhelmingly busy schedule, but will bend in order to help get things taken care of.
I/O Controls

```c
void checkKeyboard (bool& go);
void GLFWCALL mouse_click (int button, int action);
```

**Keyboard:**
The `checkKeyboard` function is to enter a key to make a selection. When the user presses the key:

- **Q/q** Exit the game
- **1** Start the game with level 1
- **2** Start the game with level 2
- **3** Start the game with level 3
- **4** Start the game with level 4
- **5** Start the game with level 5
- **6** Start the game with level 6
- **M/m** Go to Menu
- **S/s** Stop the game

**Menu Page:**

![Menu Page](image)

**Last Page:**

![Last Page](image)
After testing, the options Q/q, M/m, S/s, and 1 worked properly. The rest of the options (2, 3, 4, 5, and 6) didn’t do anything because we haven’t design the level of the game yet. Only level 1 of the game is available for right now.

Code:

```c
void checkKeyboard ( bool &go )
{
    if(!WIN)return;

    if (glfwGetKey ( 'Q' ) == GLFW_PRESS || glfwGetKey ( 'q' ) == GLFW_PRESS )
    {
        go = false;
    }
    else if ( glfwGetKey ( '1' ) == GLFW_PRESS && screen->number == 1 )
    {
        reset ();
        loadLevel(1, ai, shapes, clientLoc, serverLoc, leftDistance, btmDistance, cellDim, rows, cols);
        screen->number = 2;
    }
    else if ( glfwGetKey ( '2' ) == GLFW_PRESS && screen->number == 1 )
    {
        //screen->number = 2;
    }
    else if ( glfwGetKey ( '3' ) == GLFW_PRESS && screen->number == 1 )
    {
        //screen->number = 2;
    }
    else if ( glfwGetKey ( '4' ) == GLFW_PRESS && screen->number == 1 )
    {
        //screen->number = 2;
    }
```
The mouse_click function is to click the cell in the grid to rotate the object. The user is only allowed to click the left button of the mouse. Then the object will perform a clockwise rotation to the desired position. Therefore, when the user clicks the right button, the cell will not perform an anticlockwise rotation.

After testing, the mouse_click function worked properly.

Code:

```c
void GLFWCALL mouse_click(int button, int action){
    if ( action == GLFW_PRESS && button == GLFW_MOUSE_BUTTON_LEFT){
        if(currCell1 || currCell2->rows*cols > WON || screen->number==2) return;
        shapes.at(currCell1-1)->rotate();
        for(int n=8; n<shapes.size(); n++){
            shapes.at(n)->isInWinPath=false;
        }
    WON=ai.userWon(shapes, serverLoc, clientLoc);

        if(WON){
            screen->nextScreen=3;
            pthread_t thread;
            pthread_create(&thread, NULL, startThread,
                (void*)screen);
        }
    }
```
Algorithm
The algorithm makes sure that the wires are connected every terminals to the main server by clicking the cell in the grid.

Rules:
- There are three different kinds of objects: The server, terminals, and wires.
- There is only ONE server, and the user is unable to rotate it.
- There are MULTIPLE terminals and wires. The user is allowed to rotate them.
- The user can rotate the objects clockwise only by clicking the left button of the mouse.
- There is only ONE solution for the game.

After testing, the algorithm worked properly.

Code:
http://www.cs.csubak.edu/~tkong/cs335/Final/Algorithm.cpp

Problem
The game is only compiled and ran in one member’s laptop. Other members cannot compile it in sleipnir or other software. Even though we include all the libraries and the tools we need, we still have many error messages that prevent us to compile the program. Right now, each member of our team is trying to compile it. Hopefully we can find out the solution.