Tux Bomber

COMP 0x131B

Table of Contents

Project Teams 3

Game Logic 3

Graphics 4

Networking 5

User Guide 6

System Requirements & Dependencies 6

Compilation 6

Game-play 7

Project Overview 15

General Server Overview 15

General Client Overview 16

Game Logic 17

Graphics 18

Networking General 19

Networking Server 20

Networking Client 22

Game Description 24

Movement & Combat 25

Destructible Blocks & Non-destructible Blocks 26

Challenge Analysis & Solutions 27

Analysis of the Challenge 27

Challenges & Solutions 27

PROJECT SCHEDULE 28

First & Final Integration 29

SUMMARY & COMMENTS FROM THE EXECUTIVE 30

Matt Creamer – Project Resource Manager 30

Colby Last – Game Logic Leader 32

Curtis Anderson – Graphics Leader 34

David Young – Networking Leader 36

APPLICATION TESTING 37

ACKNOWLEDGMENT WHERE DUE 38

Project Teams

Throughout the term of the project, group leaders were responsible for the output and performance of their respective groups. Additionally, they were responsible for facilitating inter-group communication and feature requests as well as ensuring that members adhered to project deadlines and that work is being delegated fairly within their respective groups.

Project organizational and technical leaders had to work closely with each group leader and the project members as a whole to ensure that the entire project would succeed. Co-ordinating groups effectively and spearheading overall project issues and challenges were their primary tasks. That is, in addition to the mountain of monkey code work that they had to do.

Game Logic

The game logic group was responsible for the overall behaviour of the application with respect to how players interact and game rules. Everything that a player tries to do within the game would be funnelled through game logic before being passed onto the network or graphics.

One can consider game logic to be the heart of the application, where nothing would happen without explicit permission from game logic.

Team Members:

- ★ Alin Albu
- ★ Henry Berrisford – Project Technical Manager
- ★ Mitesh Lad
- ★ Colby Last – Group Leader
- ★ Brendan Neva
- ★ Clay Sayer
- ★ T.J. Thind

Graphics

The graphics group is the end-point for all communications through game logic. The end-product of the graphic group was to put a face to all the interactions that occurred within game logic. Graphics was entirely responsible for communicating a user's interactions within the GUI to game logic and translating any information that came back from game logic into interactions on-screen.

Basically, graphics adds all the fun and pizzaz to a game that would otherwise be a rather boring affair of grid-points and numbers in the terminal.

Team Members:

- ★ Curtis Anderson – Group Leader
- ★ Jaymz Boilard
- ★ Matt Creamer – Project Resource Manager
- ★ Jerrod Hudson
- ★ Max Wardell
- ★ Matt Woodske

Networking

The networking group was entirely responsible for communication between the server and clients of this network game. More specifically, networking dealt with communication between client-side logic and server-side logic.

Whenever a user performs an action within the GUI, it would be sent to game logic, which then would call the appropriate networking function to transport the needed data across the network.

Team Members:

- ★ Kyle MacDonald
- ★ Steffen L. Norgren
- ★ David Young – Group Leader
- ★ Eddie Zhang

System Requirements & Dependencies

The system requirements of this application may vary depending on individual Linux or BSD distributions, however, there are a few minimum requirements that need to be met on any of these systems.

★ A relatively recent Linux or BSD running kernel.

★ libsdl 1.2+

★ libsdl-image 1.2+

★ libsdl-sound 1.2+

★ libsdl-gfx 1.2+

Additionally, in order to enjoy the multiplayer capabilities of this application, one must have network access!

Compilation

In addition to meeting the minimum system requirements outlined above, one requires the following installed in order to successfully compile Tux Bomber:

★ g++

★ libsdl-dev 1.2+

★ libsdl-image-dev 1.2+

★ libsdl-sound-dev 1.2+

★ libsdl-gfx-dev 1.2+

Once these requirements are met, you should be able to compile the application by issuing the following command from the source directory for both client & server:

make

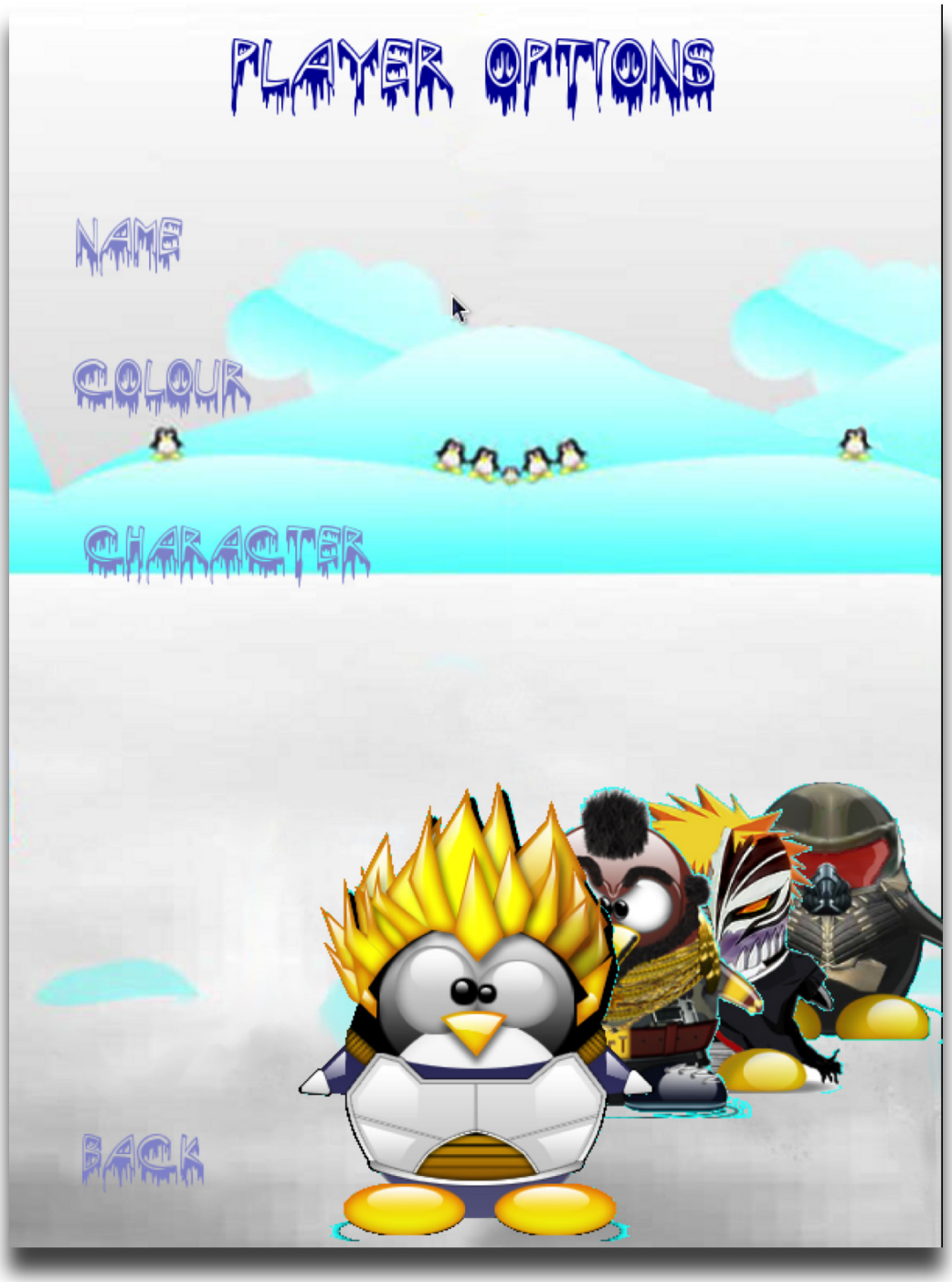
Game-play

Once the game has been compiled and loaded, the user is greeted with a rather colourful menu from which they have several options. The first option is obviously “Start Game”, which moves the user into the lobby to await one or more people to join.

From within the “Game Options” menu, the client can set the IP address for the server to which they wish to connect.

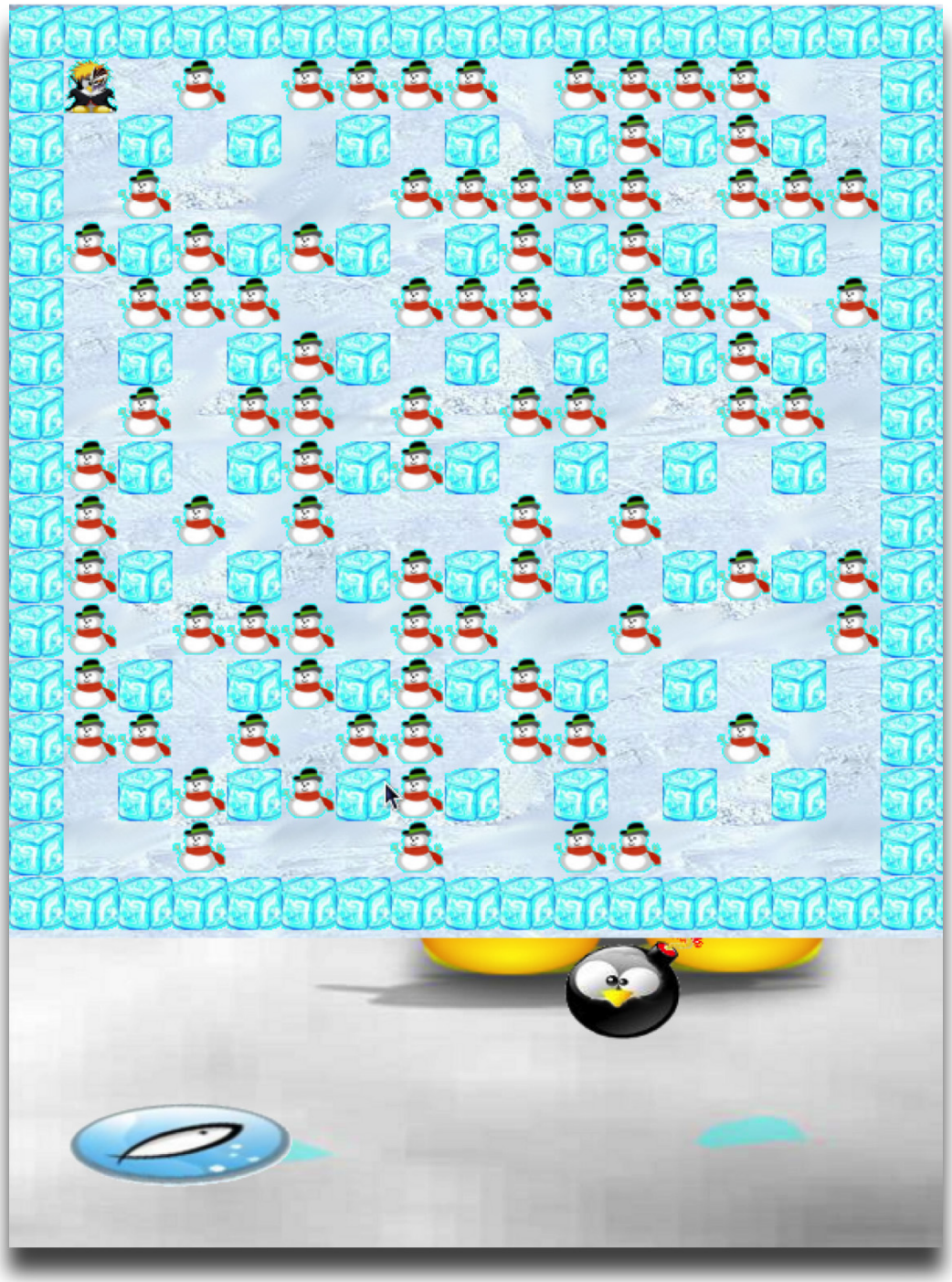
If the user wishes to customize their character name, appearance, or colour, they can do so from within the “Player Options” menu.

And, of course, of this whole sort of general mishmash scares you, you may exit the application immediately.

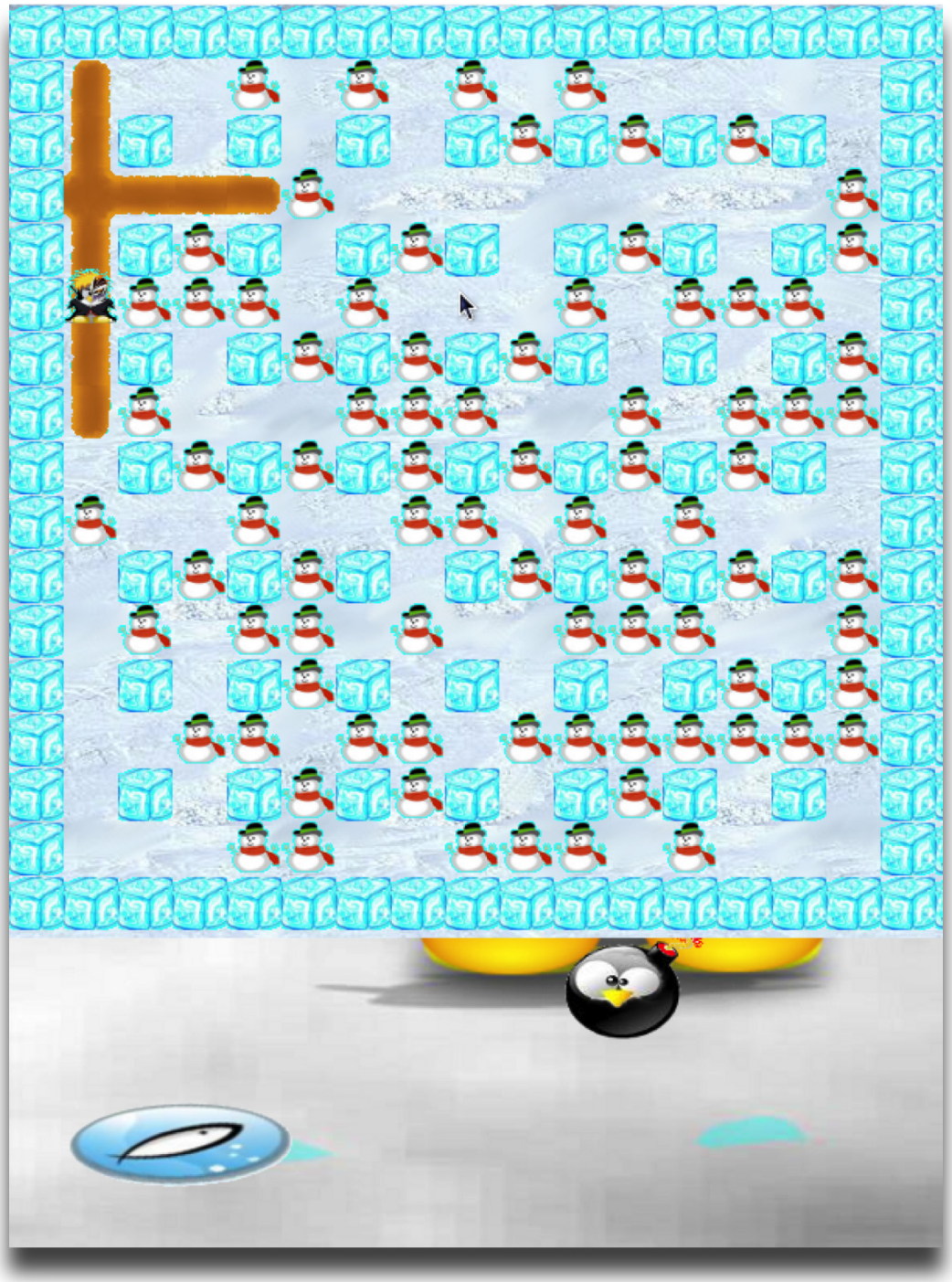


As mentioned earlier, you can change the name, colour, and appearance of your character from within the “Player Options” submenu. This can be done by clicking on the various options presented.

In order to select a different character, you need to make sure that the “Character” menu option is highlighted, and then you can cycle through the available characters by pressing the left and right arrow keys on the keyboard.



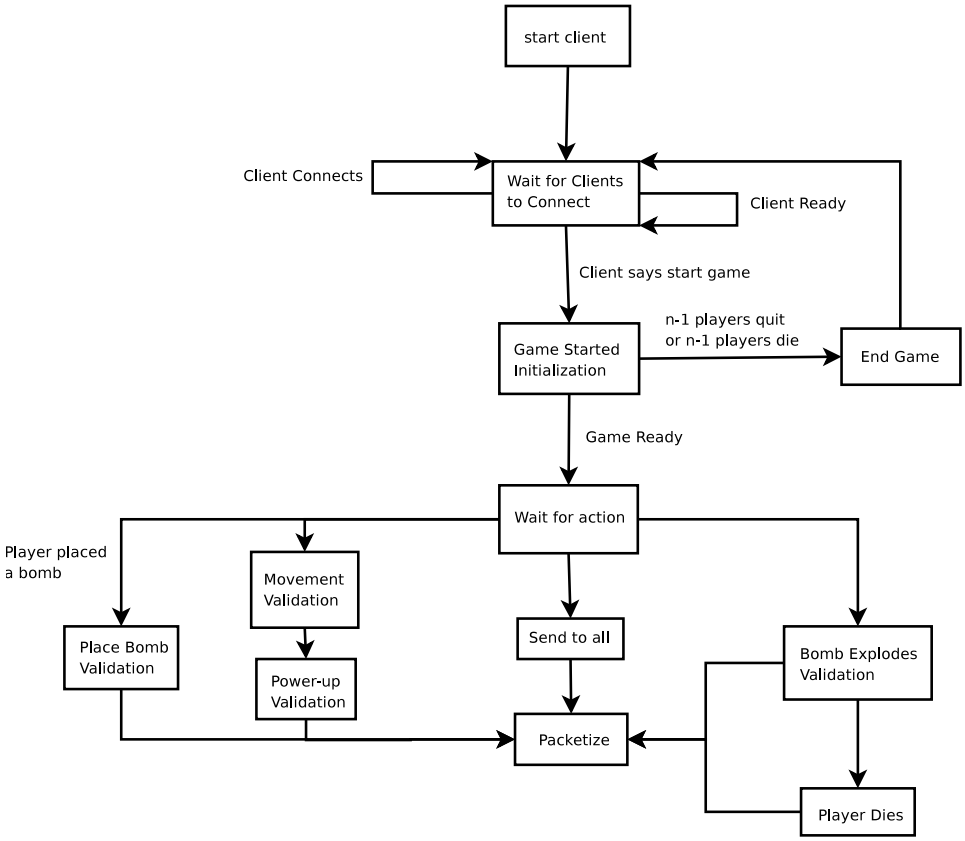
Once you have set the server IP and made any changes to your character that you may have desired, you can enter into the exiting world of Tux Bomber!



Just be careful to not blow yourself up.

PROJECT OVERVIEW

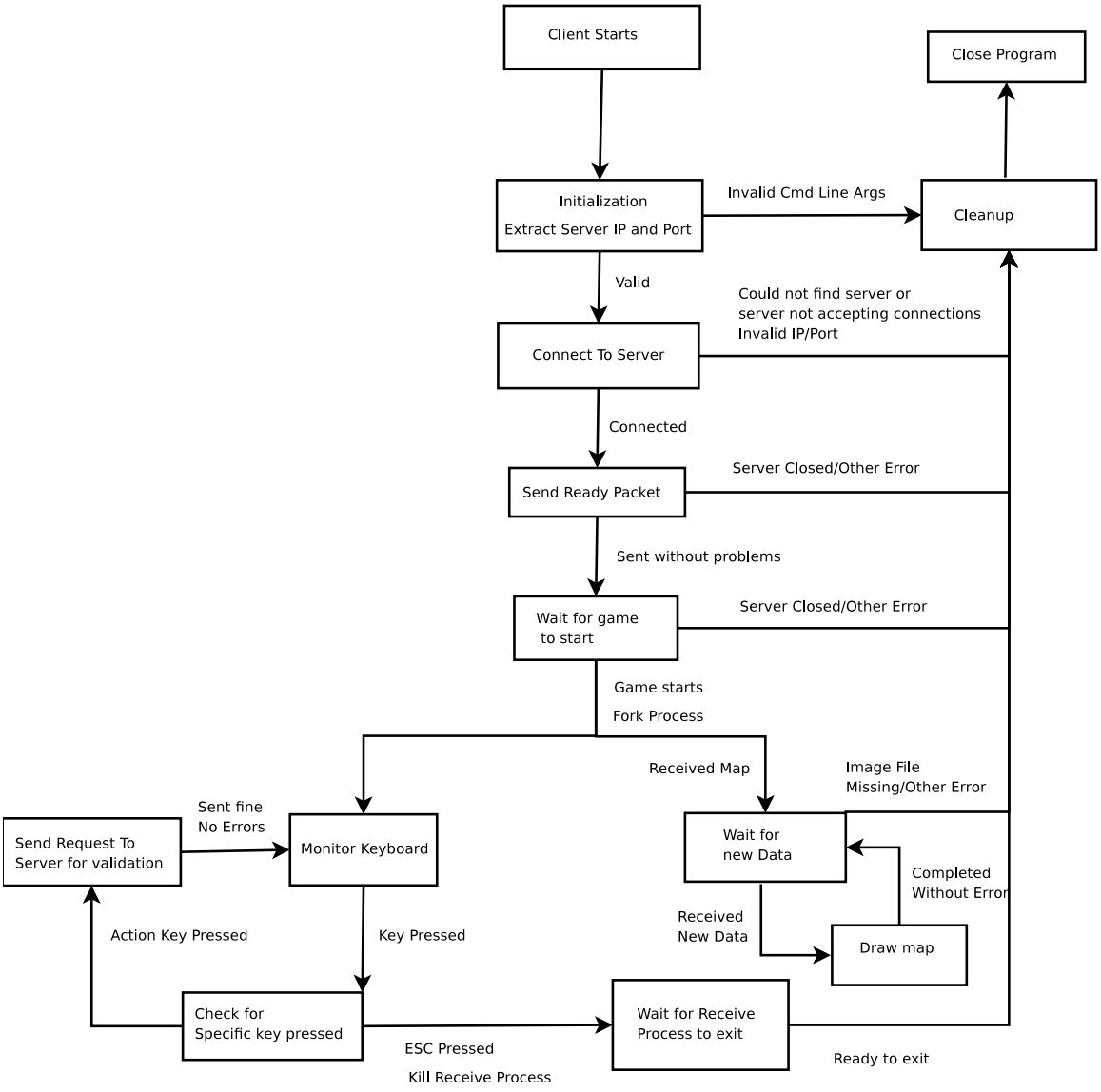
General Server Overview



In the above diagram we show how the server application relates to the various parts of our application. Basically, the server will continually wait for new client connections until one of the connected clients initiates a new game. It is at this point that the server requests a ready response from each client before continuing into an actual game.

Once the server is in “game mode” the majority of server communications revolve around game logic decisions such as player movement, bomb placement, etc.

General Client Overview



In the above state chart diagram, we outline a general connection procedure for when clients connect to the game server and initiate a new game. Initially when the client connects to the server, the server sits and waits in a “wait to start game” state where it continues to allow new connections from clients until either eight clients have connected or one of the clients initiate a new game.

Game Logic

Internal game logic was responsible for creating the code structure of the game play itself. Players and the grid upon which they exist and interact had to be first done in code. All player actions, as well as actions taken against them by their opponents, must follow the predefined rules within game logic. This includes features such as collision detection, playable areas and character movement. Basically, game logic is the central clearing house for all interactions that take place between two player within the game.

Technical breakdown:

Map creation is a randomly generated process where destructible blocks are randomly placed, but all non-destructible blocs are fixed. The map is represented by a simple two dimensional array containing numerical elements to represent players and map objects. When a map is updated, it is simply sent out to all currently connected clients.

The basic concept is that all clients are given a map that is entirely controlled by the game logic on the server. The elements that game logic explicitly tracks during game-play is as follows:

- ★ Moveable characters.
- ★ Bombs placed on the map.
- ★ Map space available in which the players can move
- ★ Destructible blocks
- ★ Indestructible non-playable game space

The protocol for what constitutes a valid event will be a collection of server event validation conditions, which include the following:

- ★ Explosions occupying the same place as a player will kill the player
- ★ After a bomb has been planted, a character cannot pass through it
- ★ When a bomb is first planted, any characters on that square may remain on that square

When a bomb is planted, a new timer is created on the server that has a timer counting down. When the timer reaches its limit, the bomb detonates and creates a blast radius. The number of bombs you can plant are represented by variables contained in the

server. When an event to plant a bomb is sent to the server, attached to that event request is your player ID. Once a specific quantity of bombs has been planted with that ID, the server will either plant or deny that request.

Graphics

The graphics engine used for this project is SDL, which is implemented in a C++ coding environment using the Eclipse IDE. A basic set of interfaces and classes have been provided with which game logic can connect and communicate incoming events. One of these interfaces is a drawing interface that enables updates of a map grid, with which player positioning is determined.

When the application is first started, the player is greeted with a main menu, which allows for a number of player customization choices and server connection options. It is a relatively basic main menu and many of the features originally planned for had to be scrapped in order to make a workable product.

Technical breakdown:

- ★ Uses SDL in a C++ coding environment using Eclipse as an IDE
- ★ Provides a set of SDL graphics interfaces that can be easily bound to game logic
- ★ Uses a map grid to determine bitmap positions and game-logic reactions
- ★ In-game menu system that includes the following menus:
  - ★ Game Options
  - ★ Player Options
  - ★ Start Game
  - ★ Exit Game
- ★ Support for customizable bitmaps for character sprites, background, and entities

Networking General

The network portion of the project implements a simple client/server design. Clients send packet requests to the server via a TCP control channel, whereas game-play takes place over UDP in order to reduce lag between the clients.

When the host creates a server, clients can join the server by way of direct IP connection. They will send a request packet detailing their player information to the server and if the information is correct, they will be accepted into the servers player list.

When the host elects to start a game round, he will send an instructions packet to each of the clients informing them that the game will be starting, as well as sending game object information to each client.

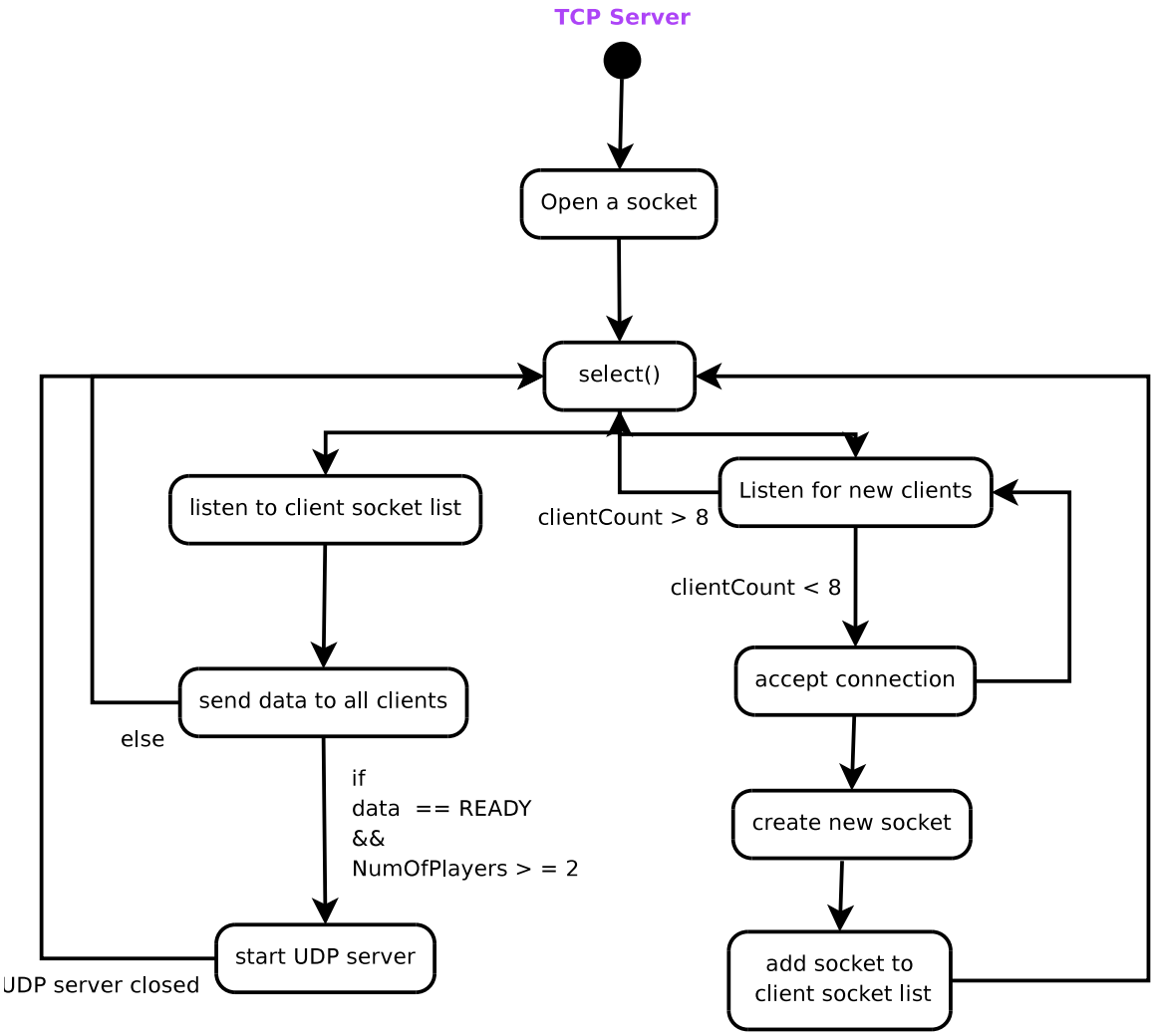
During the play phase, clients will send play requests to the server requesting permission to plant bombs, move, etc. If the server finds the request violates game rules, it will ignore the request; however, if the request is validated, it will send an update package to each client informing them of the action to be performed.

When the game is finished, the server will send a "game finished" packet informing all of the clients that the game has been completed and will include information such as round winner and player scores.

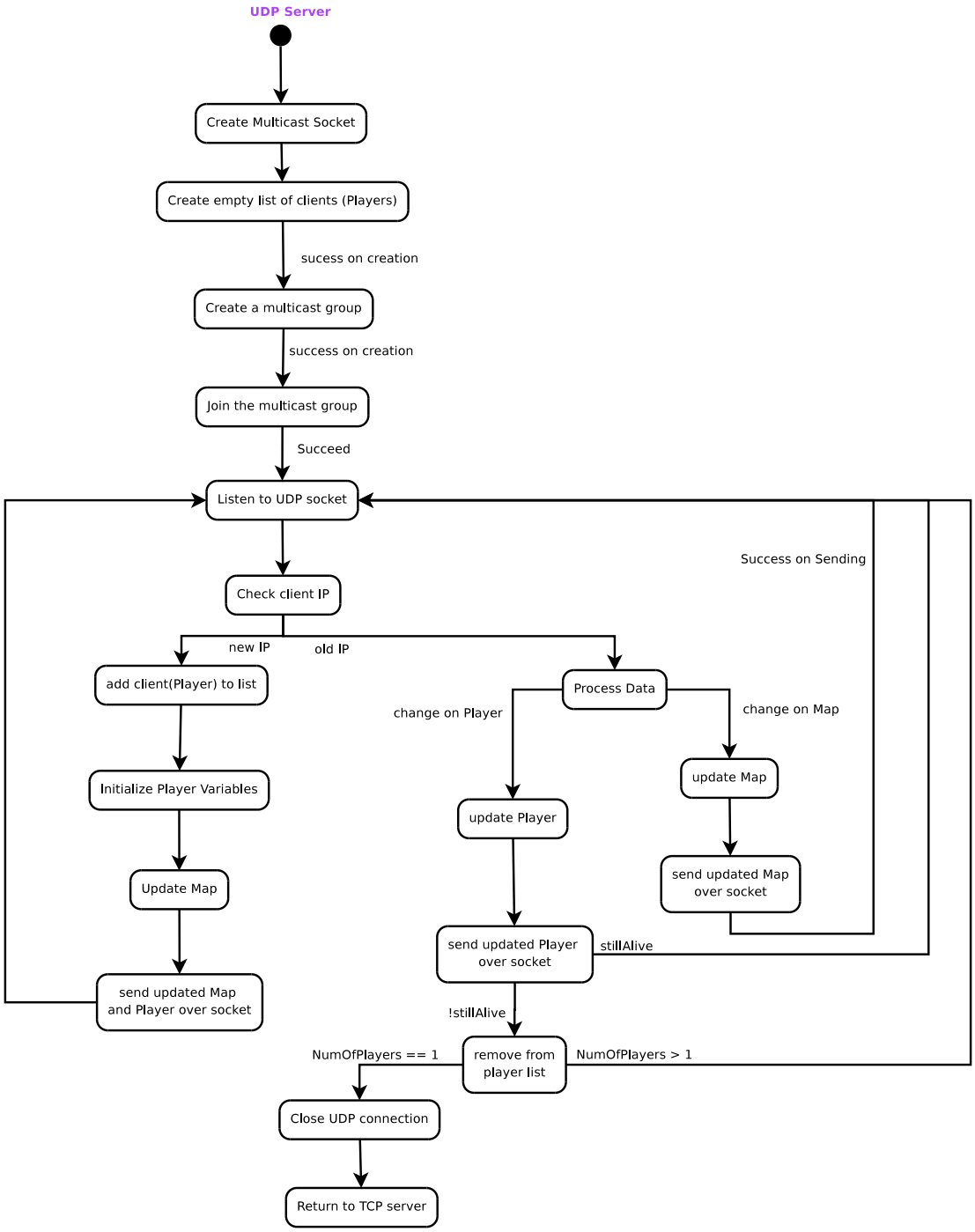
Technical Constraints:

- ★ Implements the TCP protocol for control and UDP for game-play
- ★ Implements client server model
- ★ Uses a custom designed data packet
- ★ Clients send requests, server processes, server sends instructions

Networking Server

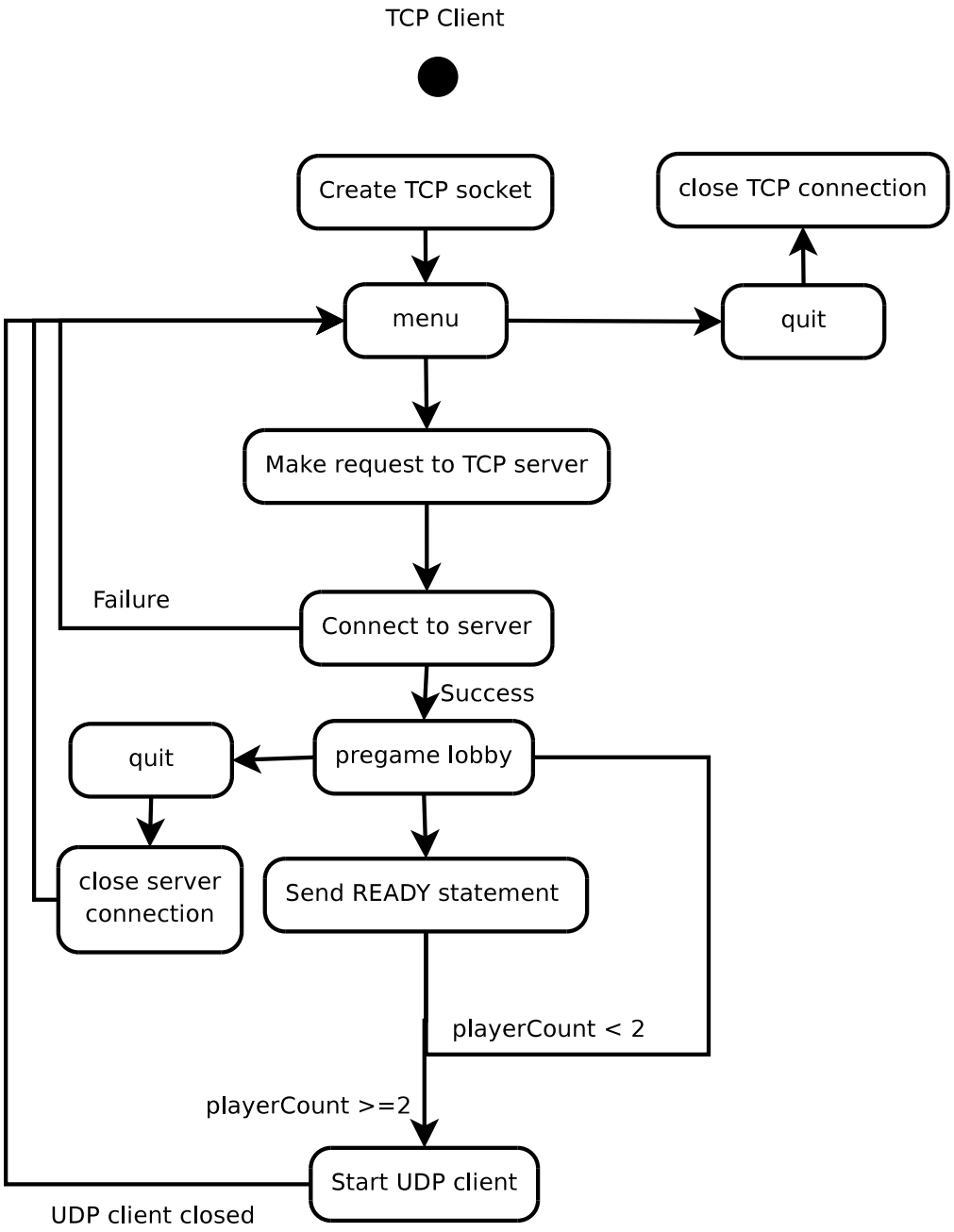


This is a simple overview of the TCP server that is responsible for being a simple control channel between the client and the server once a client has connected. Following this state chart diagram, we further outline the functionality of the UDP server that is initiated upon a new active game session.

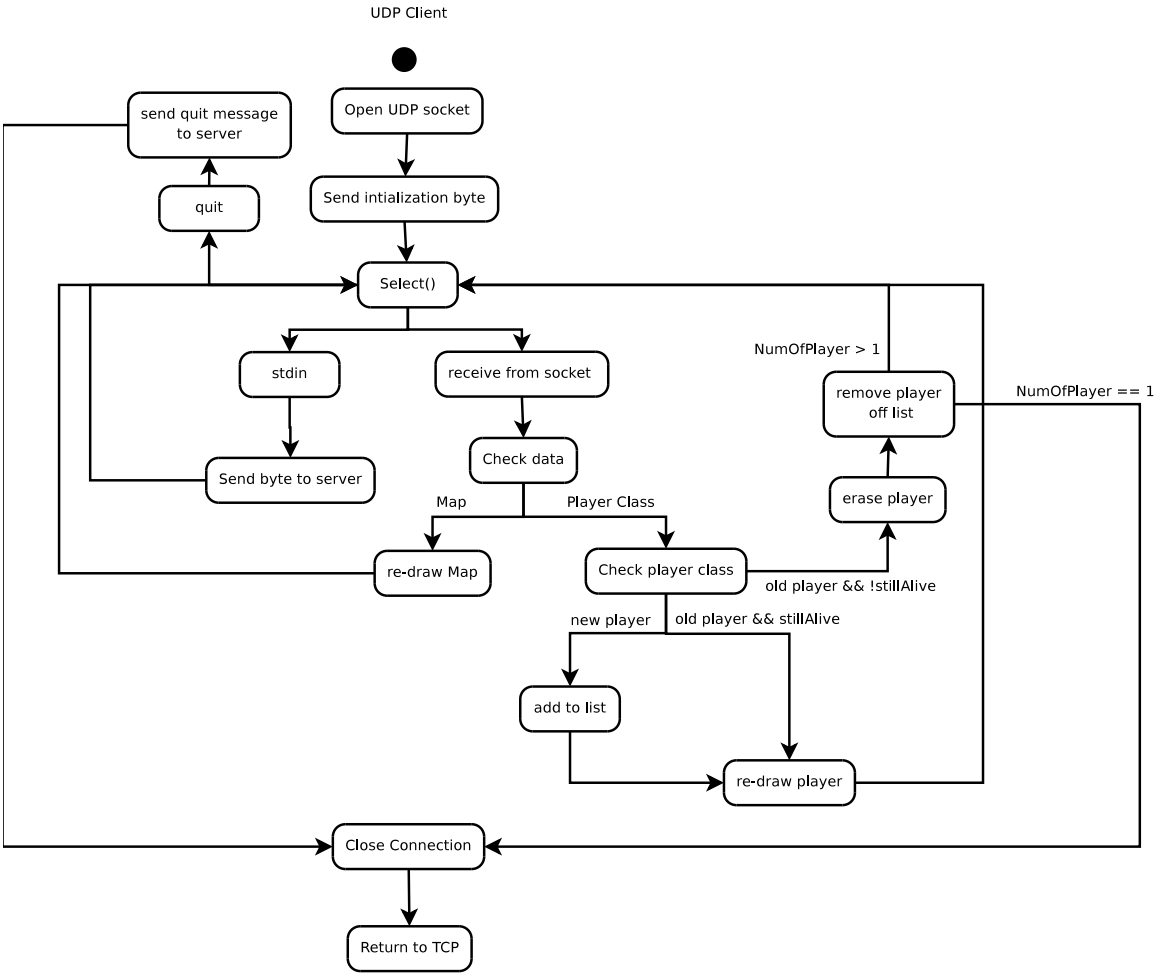


The above state chart diagram is an extension of the “start UDP server” section in the previous diagram. This basically outlines all the events that take place in order to serve a UDP-based game session with the client.

Networking Client



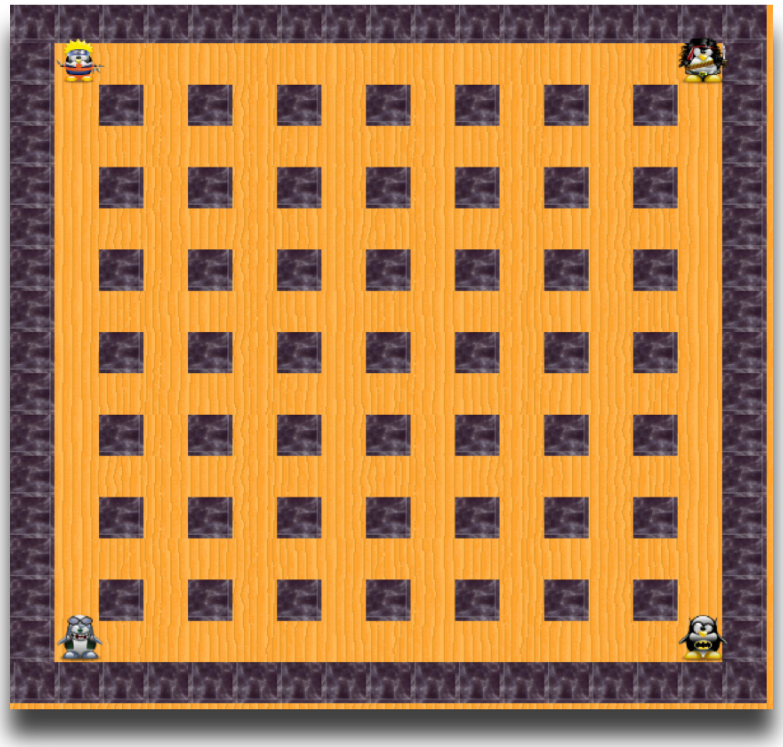
As with the two previous state chart diagrams outlining server interaction with the client upon establishment of TCP and UDP sessions, these charts deal with the client's point of view when establishing the same connections previously outlined.



This state chart diagram outlines the procedures involved with the creation of a UDP client connection to the server once a game session has begun.

Game Description

Tux Bomber is a grid-based, arcade-style, game that is based off the original Bomberman by Hudson Soft. The original game was a single-player game where the user would play against a game AI. We decided to expand upon this popular piece of vintage gaming software by adding the ability for network play with up to 8 clients. This is a classic last-man-standing type of game where the last player left alive is declared the winner.



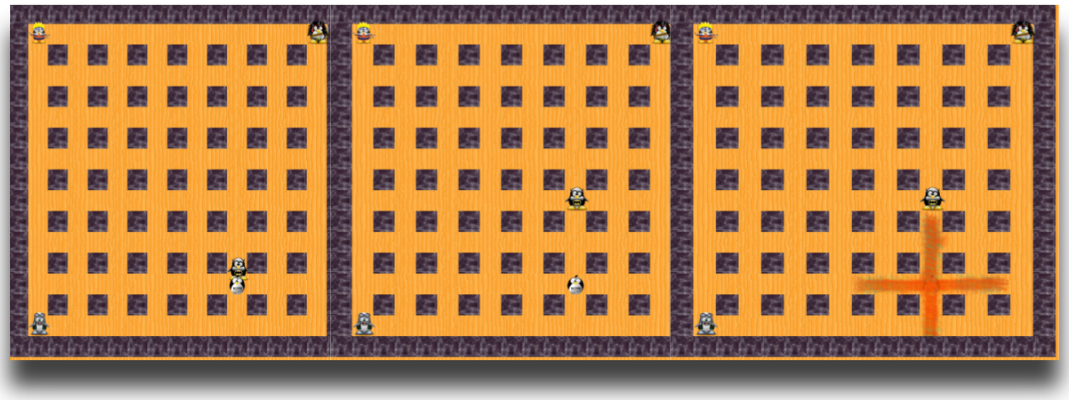
Movement & Combat

Each connected player combats each other by strategically placing bombs throughout the map in order to cause the death of their opponents. The act of dropping a bomb on the map activates the bomb's three-second timer, giving the player who dropped the bomb little time to find shelter.

When a bomb explodes, its destructive reach is only focused directly in the x and y axis. That is, anything apart from a permanent feature of the map that is along the x or y-axis of the bomb's explosive path is destroyed or killed. If a player is killed by such an explosion, they lose the round and must wait until all but one player is dead before being able to join a new round.

In the case of another bomb being caught within the destructive radius of an explosion, it will also explode, thereby increasing the possible destruction. This can possibly result in interesting scenarios where chain-reactions are possible.

Upon exploding, the respective bomb will disappear, thereby freeing the associated player to place a new bomb at their discretion.

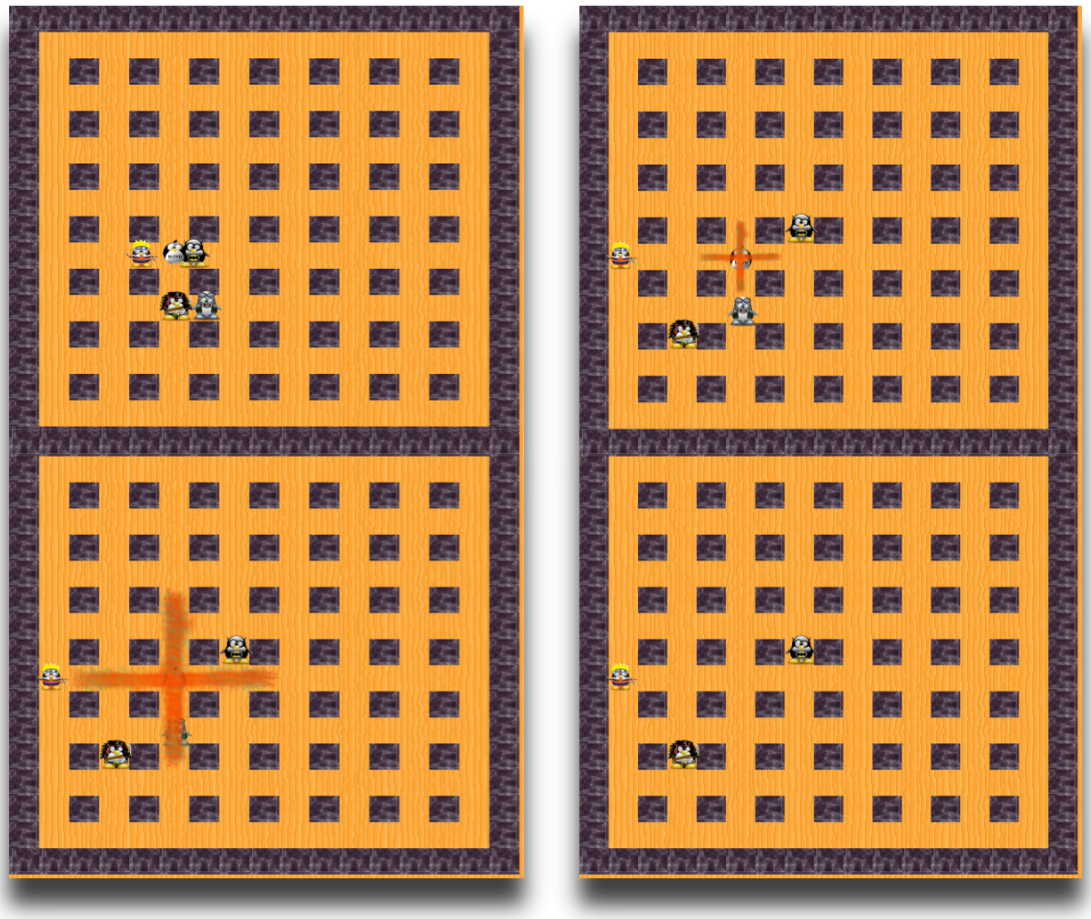


In order to add a further element of strategy to the game, players are unable to walk through other players, as well as any bombs placed on the map. This introduces the possibility for a player to trap an opponent into a situation from which they cannot escape.

Destructible Blocks & Non-destructible Blocks

The player map contains two distinct types of blocks; destructible and non-destructible. The non-destructible blocks, as the name implies, are permanent and fixed features of the map. They cannot be moved through by players, nor can bomb explosions pass through them.

Destructible blocks, on the other hand, are randomly placed blocks throughout the map, which can be destroyed by bomb explosions. However, despite being destructible, these blocks to still offer an element of protection to players, as any bomb explosion will be absorbed by the block that is being destroyed.



Challenge Analysis & Solutions

Analysis of the Challenge

As we began to work on the organization of this project, the first challenge that we encountered was that of how to break up the work into discrete units with which specific groups could be tasked.

Our first challenge was trying to determine how we could split up the project work into four working groups. The main reason we had this number in mind is simply because this tended to the number of groups used for previous year's projects in the course. After much figuring and discussion on the matter, we found that the only logical way we could split up the project work was into three discrete units; graphics, logic, and networking.

At the time of resource allocation, we assumed that the majority of the man-hours would be required in the graphics team; thus we assigned seven people to that team and put five people in game logic and five into networking.

Thus we had created our formal group compositions, each with a clearly defined task and goal to achieve.

Challenges & Solutions

As the project progressed, we found that our initial estimate for resource allocation was sorely miscalculated. The networking group had very little work to find for five people and the graphics group was encountering similar issues. Our solution to this issue was to simply reallocate resources to groups where it was needed, specifically game logic.

Unfortunately, despite this reallocation of resources, we still encountered issues with not having enough work for everyone in the class. That is not to say that everyone didn't contribute to the project. It simply wasn't feasible to split up coding assignments such that everyone could contribute to the project in fluidic manner without causing roadblocks to code completion. We basically couldn't justify splitting up tasks that could easily be handled by a single individual.

PROJECT SCHEDULE

Our original project schedule was, in hindsight, overly optimistic. We had originally planned that at the end of each three week period, each team would integrate their current work with that of the other groups in order to generate a workable prototype. Unfortunately this was not the case, as one group always seemed to be in a position of being unable to move foreword until another group provided some form of deliverables.

Regardless of this rather difficult issue, below is the original project schedule:

Iteration 1 - (Feb 3 - Feb 24)

- ★ *Graphical User Interface* - Development Kit
- ★ *Game Logic* - Collision detection and character movement
- ★ *Network Communication* - Server/Client sending and receiving data

The goal of the first iteration was to have a simple working prototype within each group, but a full integration between each groups work. For the most part, each team had been able to produce these deliverables.

Iteration 2 - (Feb 24 – Mar 17)

- ★ *Graphical User Interface* - Displaying the map and characters, and correctly changing their position as the user inputs commands
- ★ *Game Logic* - Consistent Game control
- ★ *Network Communication* - Multi-client, advanced data control

The second iteration and integration period was where the project began to seriously lag behind. It was at this point that any plans for integration between each group was pushed back until after spring break.

Iteration 3 - (Mar 17 – Apr 7)

- ★ *Graphical User Interface* - Menus, and additional visuals\audio
- ★ *Game Logic* - Actual game play (Bombs and hit boxes)
- ★ *Network Communication* - If additional time permits: In game chat window

Having learned our lesson in the previous term's final assignment, our class started out fired up, and ready to conquer this project. For the initial three weeks of this project, our class spent literally all of our time in mass debates about what project we should choose, why we should choose said project, and finally, what we need to do for the chosen project. Once Tux Bomber was decided on, and with very high hopes, our class organized a very structured and pristine schedule of incremental deadlines. The class was then broken into it's three main groups (Logic, Graphics and Network) and we gave each group's team leader the task of schedule internal deadlines. The class as a whole was given three integration deadlines in the hopes that each group would have met their deadlines. This is not what happened.

As time went on, Tux Bomber was constantly transfigured to exist within pockets of free work time. Per week, we had allocated four hours of in-class work time, and sure enough, this turned out to be much less time than was needed to complete each groups deadlines. Even with our best intentions, as more and more external assignments piled up, our progress halted. The deadline for Tux Bomber continued to feel safely, far away. Tux Bomber did not take a forefront. Further and further past the first integration period, it became apparent that we made a crucial oversight, and that we should have been integrating everything from day one. By letting each group break off into their own world, we effectively crippled our productivity when the time came to integrate. As it stands now, we have yet to have ever completed Integration period #1.

First & Final Integration

The project’s first and final full integration of all the various components really got started on the Sunday prior to the project’s due date. It was at this point that a basic integration between game logic, graphics, and networking was completed.

This initial integration succeeded in allowing for basic player movement throughout the GUI map and communicating those changes to a second client over the network. This success sparked a whole new chain of integration successes. However, during the process of this integration, the majority of the code from networking was taken apart and rewritten as needed on a trial and error basis.

Regardless of this approach, the group’s progress began to steadily accelerate to the point where the majority of our initially planned for features made it into the working application.

Summary & Comments from the Executive

Matt Creamer – Project Resource Manager

Graphics vs. Logic – Movement Type:

Directly after this project began, Logic and Graphics started creating different forms of movement, unbeknownst to the opposing team. From the onset, both groups created code for a solution that the other group was not prepared to handle, leading to the biggest breakdown of communication to come out of this project, and the greatest problem we had to face as a group.

Once this problem was uncovered, nearly half way to the project deadline, an emergency meeting took place where every class member had to cast their vote to determine the movement type to be used. Both the Logic team leader Curtis and the Graphics team leader Colby presented the class with their argument for each of their respective positions.

Logic argued to represent movement as 2D grids filled with values representing to represent players moving, bombs exploding, and walls. Using this method made game logic simpler in theory, allowing collision detection to be a simple check of the grid to see if it contained a wall or bomb. Player movements would have been sent as a value for the grid space, and the location of the grid space, thus only two changes needed to be made for a player moving one square; the value of the square moved into, and the new value of the square moved out of, to now represent an empty space.

Ultimately, Graphics won the vote, in favour of having "fluid" graphical movement instead of grid based movement, and having to send less information overall from clients to server back to clients. This pretty much tipped Logic upside down, and new methods had to be created to allow what Logic had started to mingle properly with what graphics had envisioned.

General Comments:

As a fellow student that happens to be new to project management, I found this to be a deceptively complicated project to organize evenly. The largest problem I found myself faced with, over and over, was finding jobs for people to do that did not seem like pointless "busy work". Fortunately, this class in general was fairly receptive to instruction, and did not make my job harder than it needed to be.

One further problem I encountered in this project was dealing with internal disputes within groups, or the state of the general morale of the class. When you get a large group of sleep depraved coders together, there is bound to be disputes. These disputes typically need some form of mediator, or resolution to ensure that crankiness does not take hold of the reigns and halt any progress. This was something I found myself having to deal with on a very subtle scale, whenever large groups of us were working on co-operating code.

I would say that this was a successful project, regardless of how well it scores technically. It may not seem like a triumph from an outside observer, but from the inside, I can see how much time and thought was put in to this project. I've seen these guys work for the last year, and I know what is acceptable, and what is beyond. I can safely say that this group put their heart into Tux Bomber.

Colby Last – Game Logic Leader

From the start our group was assigned with the task for completing game logic. At the beginning, this included movement, collision detection, bomb physics, explosions, and game rules. Through many changes, our group ended up doing the client, server, re-drawing, and other tasks that were not necessarily our groups responsibility.

Task Divisions:

The first few times we started working we stayed together and hammered out some essential tasks that needed to be accomplished, how they could be accomplished, and then started accomplishing them. We then broke into mini groups of two to three people to get the major parts done. We acquired Matt Woodske part way through because we lacked strong secondary coding skills. We picked up the pace and got a lot more work done.

Once we got rolling, we had a group for movement and collision detection, client functions, and server functions. Curtis and Matt Creamer did movement and collision, Jaymz and Max were on client, and Jerrod and Matt Woodske did the server.

Towards crunch time we put all of our efforts together to finish putting everything together and making sure that integration of our own functions worked, and major testing was required.

Group Analysis:

Our team members complimented each other very well. Each two man pair had a coder and an analyser. Extreme programming was the essence of our work. Aside from a few situations where breaks are more important than work, each mini team finished tasks effectively, and there were no issues coming in during our free time to come in and get essential parts completed. Our group was also able to compensate for other groups who didn't always have stuff done on time or up to par.

What we Produced:

- ★ Collision Detection
- ★ Movement
- ★ Bomb drops
- ★ Bomb explosions
- ★ Player deaths

- ★ Game connections
- ★ Accepting commands from client/server
- ★ Sending out players and maps

What didn't happen:

Our first two months of work was basically all wasted because of a design change that was not unanimous and basically nulled out all of the work that we finished. The idea of having the player in the grid made for a similar game but in the end will not be used.

Curtis Anderson – Graphics Leader

As the group leader of the graphics team with 7 members, I found that managing a team of 7 people to be the most challenging. Whenever an argument about how to build the program started, I found it took extremely long to settle any of the battles. Also having a large group of overachievers didn't help either, everyone has an idea and refuses to listen to anyone else.

Having a group of overachievers made some things easy and others like planning sessions very difficult. Everyone wanted to do the fanciest thing possible, Everyone went home got something completely unplanned for done then demoed it to the rest of the group instead of what was planned for the regular meeting. Most of which couldn't be finished, and some of which gave false impressions to the other groups about how the project was going to be developed which caused problems later in development.

Our original plan was to build a program which could support up to 8 players. After joining the game, a player may

- ★ Upon entering the game the user is presented with a GUI
- ★ Players would be able to chose from a wide array of player models
- ★ Players can select a name and name colour
- ★ Connect to a GUI Server
- ★ Join a client lobby which contained game information
- ★ Wait for host to start
- ★ Once game started the server would generate a random map
- ★ In-game the client can plant bombs
- ★ Bombs can explode in a + shape and each prong would be cut off at obstacles
- ★ Players die when they touch the fire produced by a bomb

A linux theme was produced for this game which heavily emphasized tux the penguin. Alin also came up with a winter theme to compliment tux.

Because we had so much experience with network programming and C++ programming we had a very good idea of what we were capable of and were able to implement most all of our ideas; however, not all of our ideas could be implemented the way we wanted but most of them were implemented.

★ Fire could not be implemented separately from the map draw function. Instead it is drawn as the map is drawn. Whenever a fire square is loaded, an empty square is loaded first then fire drawn over top of it.

Commentary:

I found that other groups were trying to underachieve because all of the overachievers were stuck on our group. This caused some very strange solutions to be presented which I fought and caused much delay in the project as a whole.

Other groups fell behind during the last couple of days of the project and had to go all out at the end to catch up. I was worried that our code wouldn't work in the big test and grew impatient waiting for the other groups to have demo code. Other groups shocked me when they told me they built large sections of code without ever even compiling it late in the project. Other groups had to rebuild their code from scratch because of a switch from TCP to UDP. None of the other groups had commented or created any design work for any of their code then expected us to build it for them because we were done by then.

Challenges:

Our only challenges came from building the menu, the map class, and a last minute error that was discovered during game-play which would have threatened the project had we not solved it in time...

The menu was a challenge simply because everyone wanted an extremely fancy menu. You will see what we mean when you play it. That got fully implemented.

The map class was a challenge because the people in charge of its development didn't test anything that they built before they claimed it was “almost done”. Two weeks later it was completed.

The last minute error that threatened the project was created by an old version of the player class which continuously reloaded image files without closing them or reusing them. When our player moves at 3 pixels a movement, we would end up loading over 1000 files into memory before the program would crash in a matter of seconds. This bug was quickly crushed.

David Young – Networking Leader

I think that generally, my team did a fairly good job. Early on in the project, we got a decent amount of work done and had very little to do afterwards. As requirements changed, we worked to accommodate those changes. Near the end of the project, there was a major decision to switch from TCP to UDP, a decision that didn't seem to be that big of a deal work-wise, but implementing and integrating it with everyone else's code proved to be more problematic.

What was originally planned for the networking group, we accomplished for the most part. However, design changes kept coming in from other groups, and we had to modify things to compensate for new data formats being passed across the network.

In the beginning, we thought that it would be possible to do the entire game in TCP, but we eventually realized that retransmissions might put some players behind others. So we decided that UDP would be a better fit for the actual game-play, while still keeping TCP for control information such as player connection and game-start alerts. Other than what I've mentioned above, we didn't really make any design decisions. Everything was a hack to get things to work with the latest requirements we received from other groups.

We had initially created a set of functions to make networking tasks more readable and easier to accomplish. Unfortunately, other groups didn't wait for us to implement what they wanted before they started testing their code, so they created their own versions of networking code. When we started to integrate everyone's code into a final project, there was no time for us to make everything into neat and organized groupings of functions.

Application Testing

Due to the methods used to arrive at the final stage of this project, our entire testing procedure consists of an iterative trial and error process. Basically we've been incrementally adding features from our existing cove base and testing it incrementally.

For the most part this has resulted in having to rewrite a substantial portion of the code that we've accumulated over the term of this project. The major reason for this current state of development and testing us simply due to poor inter-group communication and a general lack of design work to guide the project.

Unfortunately this means that any in-depth testing has been impossible at this stage and certain bugs in the code can show up later on throughout the integration and implementation process.

Despite these shortcomings, the project as a whole has gone from a questionable base of incompatible code, to a fully functioning network game application: Tux Bomber.

Acknowledgment Where Due

Thanks for being a great teacher, Aman! I'm going to pass out now.