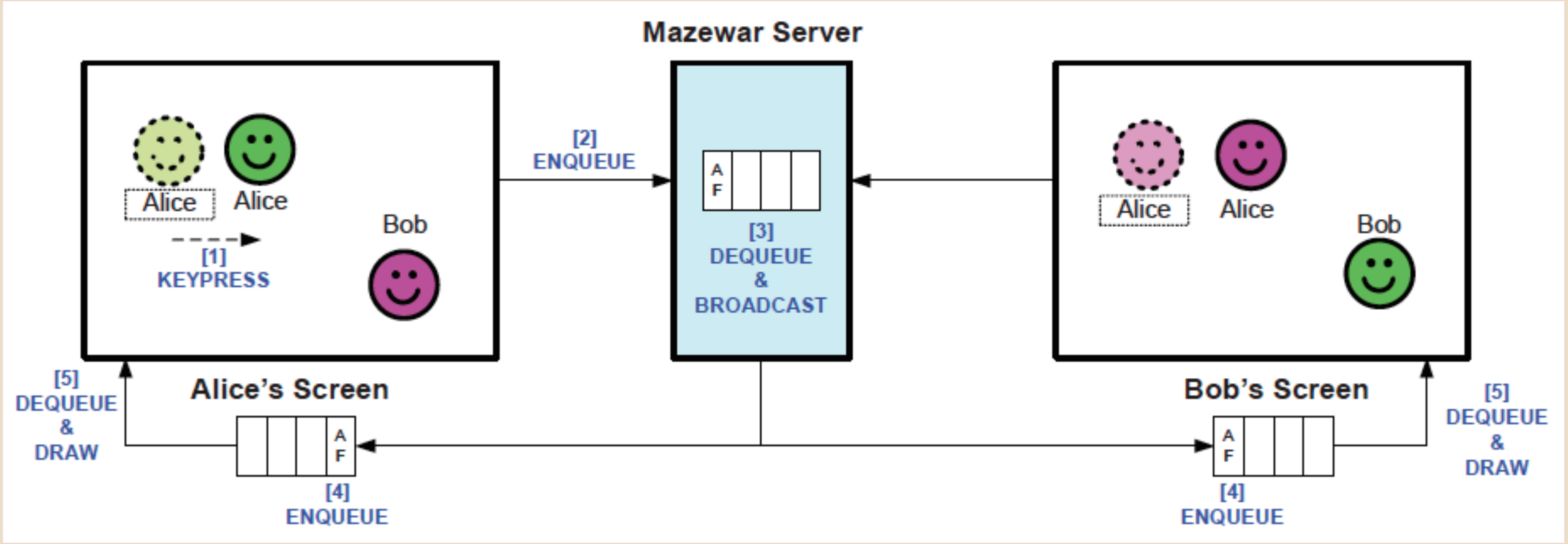
Mazewar (Lab 2) Tutorial

ECE419

Agenda

- Code Structure
  - what you need to do
- Game Initialization
- Game Play
- Applying the Patch
- Questions
- Ali: Fault Tolerance

Game Overview



Architecture

Server (Need to create Files)

Client Files (Possibly need to modify):

Mazewar.java: Creates game

*Client.java: All the different clients

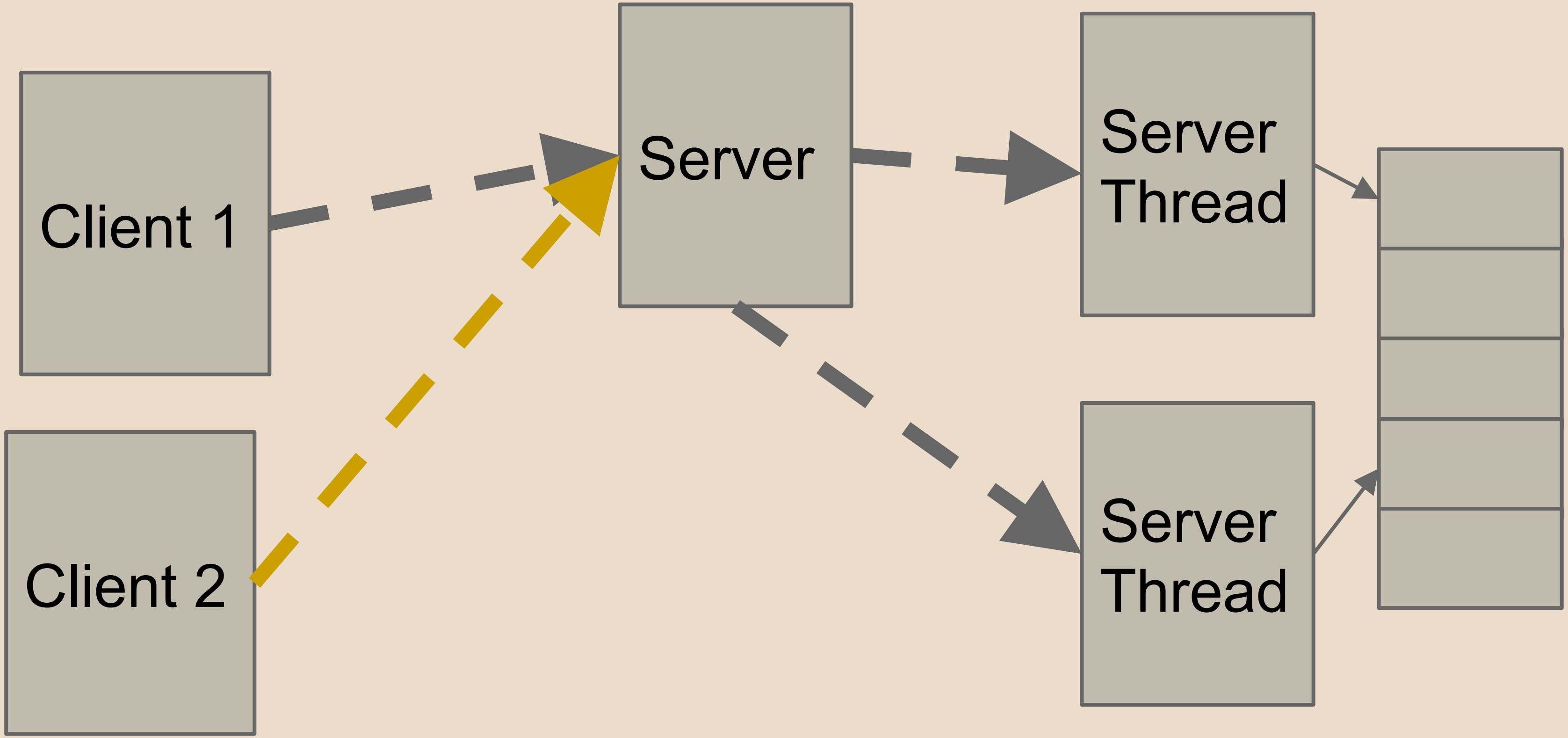
Maze.java

MazeImpl.java

Mazewar Server (High level)

```
public class MazewarServer{
    //enqueue - multithreaded since
        //many clients
  //dequeue and broadcast
}
```

Server and Server Threads



Mazewar Server

```
public class MazewarServer{
        //Error checking
    //Declare Data Structures
        ServerSocket s = new ServerSocket(3000);
    new MazewarBcastThread().start();
    while(true)
                //listens and enqueues
        new MazewarThread(s.accept(),...).start();
}
```

Server Thread Class

```
public class MazewarThread extends Thread{
  public MazewarThread(Socket s, <<other
args>>){...}
    public void run(){
        while(true){
            //read packet
            //process
            //enqueue packet
```

Alternative Thread Declaration

```
public class MazewarThread implements
Runnable{
     public void run(){
          //
  }
}
```

Client (High Level)

1) Send Events

2) Receive Events

```
while(true){
        //readpacket
        //enqueue-packet
        //execute event on appropriate client
}
```

Game Overview

Initialization

1) Create players as required

Game Play

1) Event happens on client (e.g. Bob moves left)
2) Client sends events to server
3) Server broadcasts events
4) Client executes event

Player Creation

Players are instantiated in **`Mazewar.java`**

```
guiClient = new GUIClient(name);
maze.addClient(guiClient);
```

Creating Remote players:

```
RemoteClient r = new RemoteClient(“foo”);
maze.addClient(r);
```

Client Events

see **`GUIClient.java`**, function **`keypressed`**

```
public void keyPressed(KeyEvent e) {
      ...
   // Up-arrow moves forward.
   else if(e.getKeyCode() == KeyEvent.VK_UP)
{
            forward();
     ...
```

Sending Events to Server

`GUIClient.java`, function **`keypressed,`** is where keystrokes are recorded.

//*Send them to the server here*.
//Do not call action method e.g. forward().

Client-Server Comm: Serialization

Need to serialize data; create a packet class (MazewarPkt)

```
{...
    public static final int UP = 1;
    int event;
    int player;
...}
```

Client-Server Comm: Sending Pckts

```
ObjectOutputStream sender = //initialize
...
MazewarPkt p = new MazewarPkt;
p.event = 1; //i.e. UP event
p.Player = 1; //i.e. Bob
sender.writeObject(p);
```

Client-Server Comm: Receiving Pckts

listens for incoming packets

```
ObjectInputStream receiver = //initialize
...
MazewarPkt p = new
    =(MazewarPkt)receiver.readObject();
```

Receiving Events from Server

```
MazewarPkt p;
while (true){
     p = new
   =(MazewarPkt)receiver.readObject();
     //enque packet
}
```

Moving Players

Both Remote and Local Client inherit from Client Class.

Client has `forward()`, `backup()`, `turnLeft()`, `turnRight()`.

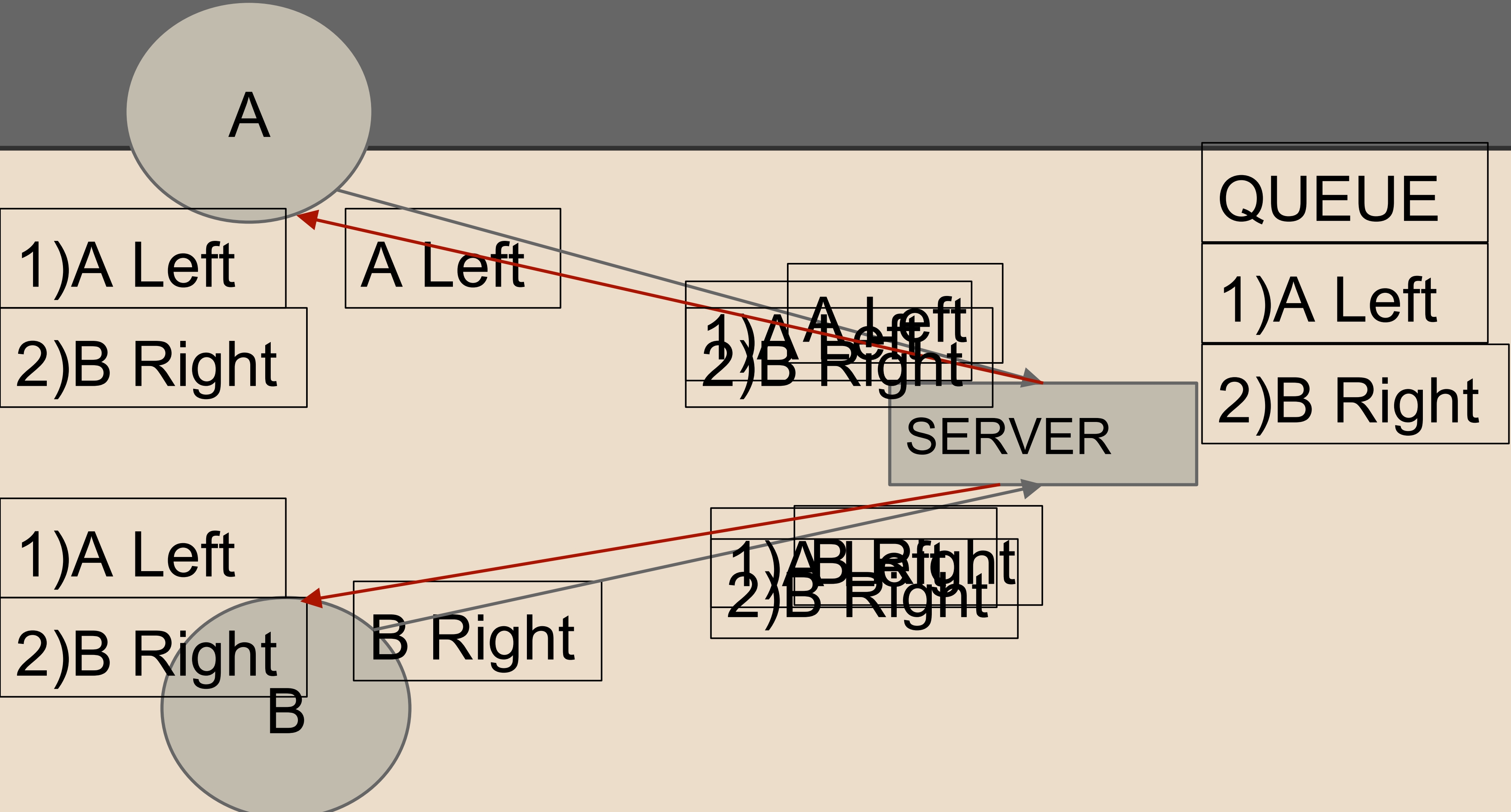
Call these methods when events are dequeued by client (this is what `keypressed` is doing too).

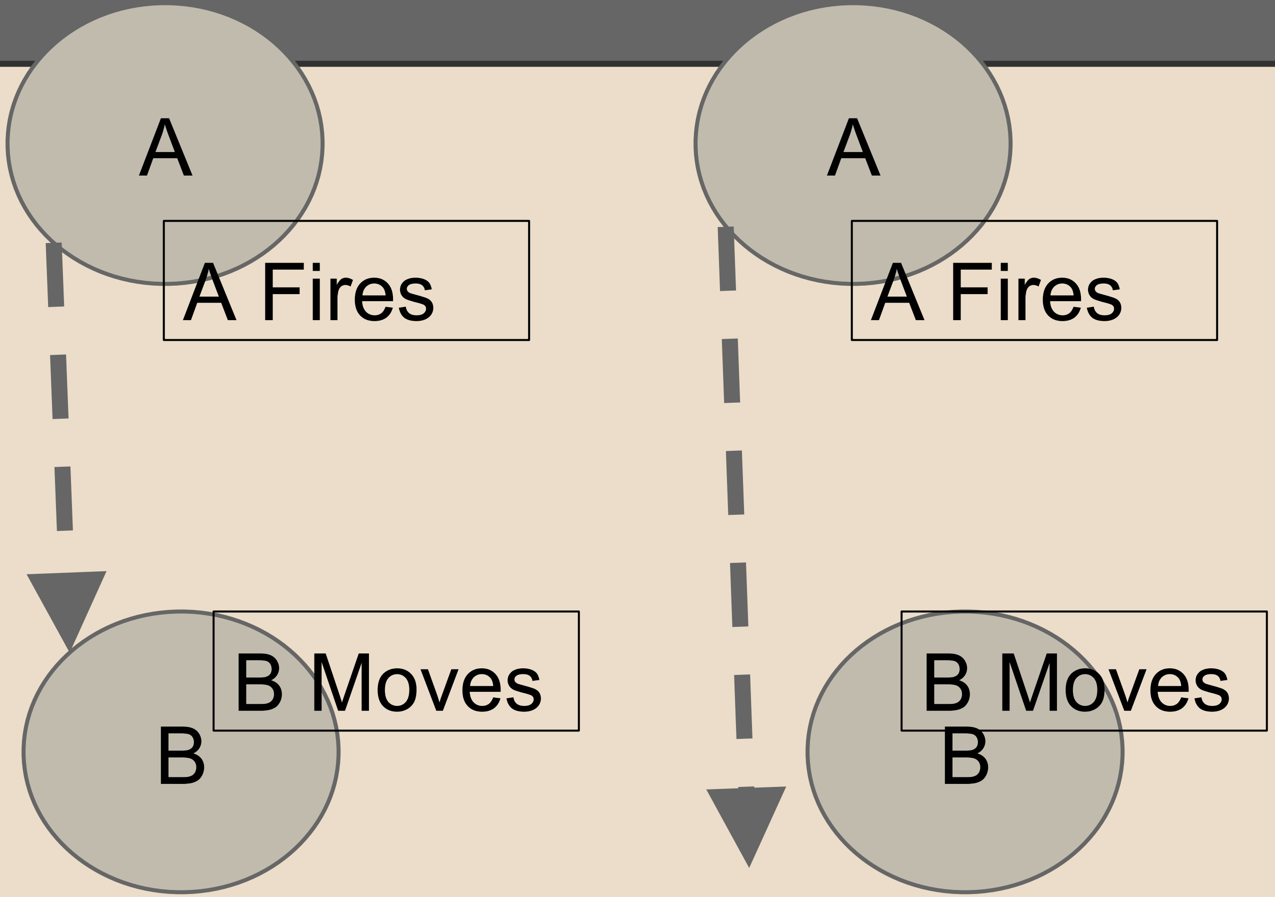
Known Issues

1) Glitch with player orientation- when player spawns, the orientation is displayed incorrectly key presses many times (left or right) will correct it
   a) Do manually, or programmatically

Known Issues -2

Synchronization Issue with Missile:
-Missile movement is not atomic
-there can be interference from other events, leading to inconsistency
-i.e. the event queue and the missile action are not synchronized with each other, their interleaving can create inconsistency


A
1)A Left
2)B Right
A Left
A Left
1)A Left
2)B Right
QUEUE
1)A Left
2)B Right
SERVER
1)A Left
2)B Right
B Right
B Right
1)A Left
2)B Right
B



Since *fire* event not sent to server, there is no ordering.
B move event might happen before or after bullet reaches B.

Fix: Synchronize Missile Tick

Currently, Missile movement being controlled in `MazeImpl.java, run` method.

1) Client: Replace `run` with `missileTick` method that moves missile every fixed period (`run` does this) so that it is synchronized with the rest of the events.

`Run`: Modifies `projectileMap,` a map of projectiles and their positions

1) Server: Use a server separate thread that adds a “missile tick” event every 200ms

Mazewar Server

```
public class MazewarServer{
        //Error checking
    //Declare Data Structures
        ServerSocket s = new ServerSocket(3000);
    new MazewarBcastThread().start();
    new MazewarTickerThread().start();
    while(true)
                new MazewarThread(s.accept(),...).start();
}
```

Server: Ticker Thread

```
public class MazewarTickerThread()extends Thread{
    public void run(){
        //Enqueue tick event
        try{
            Thread.sleep(200);
        }catch (InterruptedException e) {
         e.printStackTrace();
        }
    }
}
```

Achieving Concurrency

Use concurrent data structures- Blocking Queue interface

Synchronization methods

Synchronized statements, i.e. with locks and semaphores.

Compiling and Running

-A Makefile is provided. Will need to modify Makefile and Makefile.dep when you add your own files.
-Modify: ${ECE419_HOME} and ${JAVA_HOME}) if testing at home.
-./run.sh

Tips

-working in groups of 2, so use version control (Bitbucket provides free private repos for Git)
-Understand how the code works, you'll be able to debug better.
-Many parts of the code (queueing events) will be reused for Lab 3. Write your program so that it is easy to modify.

Tips

Lab 3, will be similar except p2p (can reuse client queuing code)

Ideally, no centralization

- -lightweight centralization (penalty)

Bonus points for:

- fault tolerance
- dynamic joins
- bots