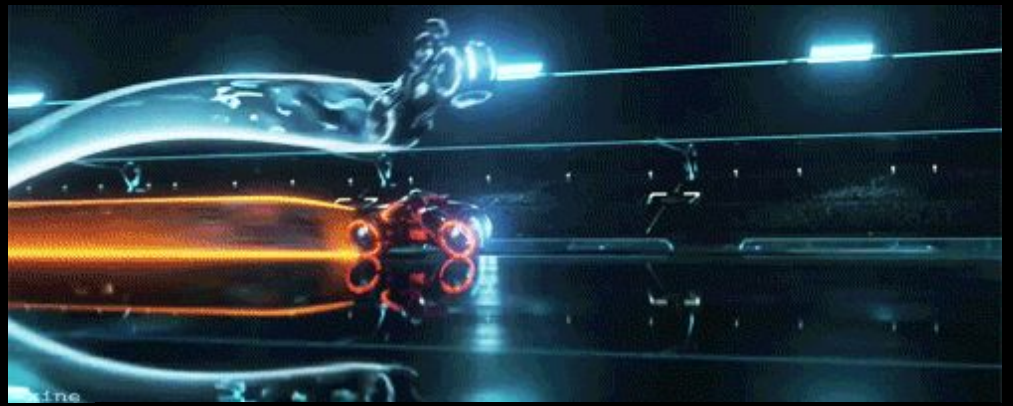
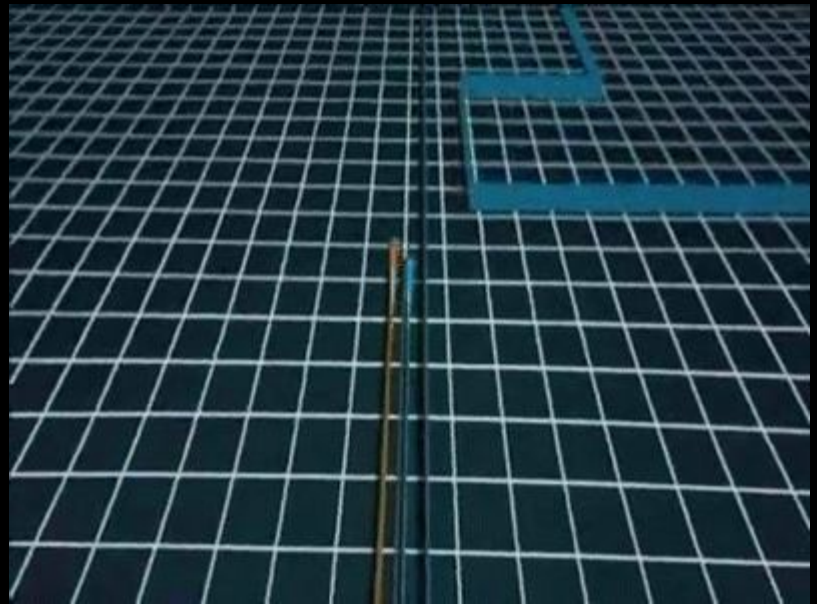
RELIABLE TRON

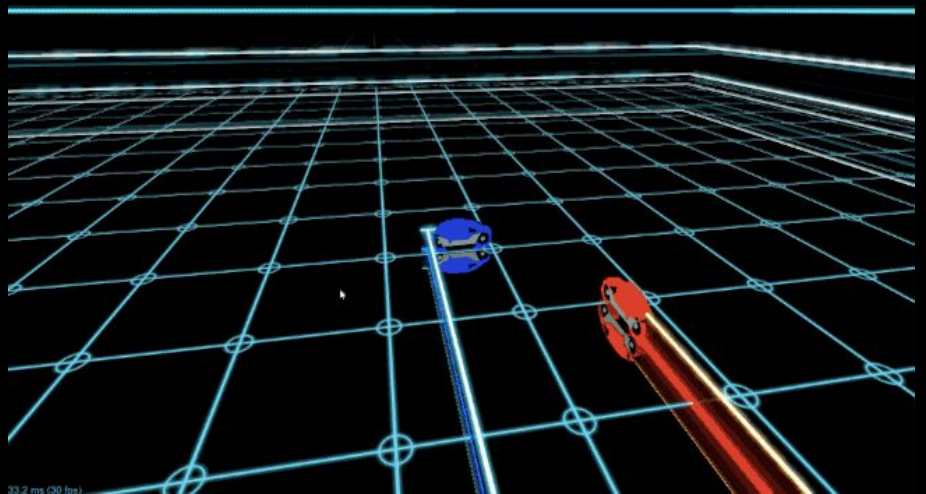
A Game Proof of Concept & Reliable Messaging

By ZJ Lu and Frederick Morlock

INSPIRATION



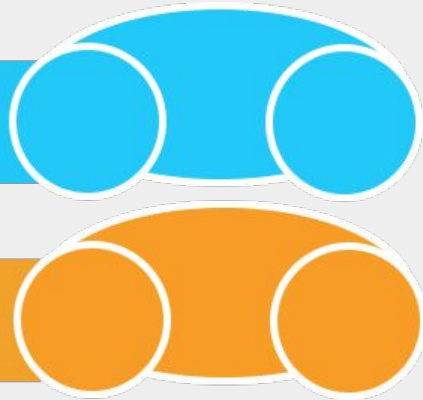




DESIGN CHOICES

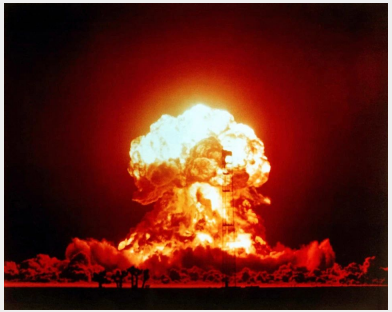
Why a proof of concept?

1. We aren't game designers
2. Not completely free of bugs…
3. Easy to "hack"
4. Latency!



DESIGN CHOICES

Why Reliable Messaging?

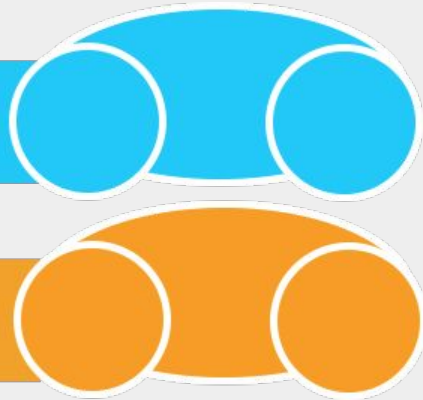


DESIGN CHOICES

Why Pinpoint Method?

1. Error Correcting
2. “Offline”

HOW DOES IT WORK?



HIJACKING THE CHAT SYSTEM...

Creating a main game loop

```
pygame.init()
FPSCLOCK = pygame.time.Clock()
DISPLAYSURF = pygame.display.set_mode((gc.WINWIDTH, gc.WINHEIGHT))
FPSCLOCK = pygame.time.Clock()
SPLASHSCREEN = pygame.image.load('SplashScreen.jpg')

pygame.display.set_caption("Reliable Tron")

WORLD = World(DISPLAYSURF)
while self.sm.get_state() != S_OFFLINE:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    DISPLAYSURF.fill((0, 0, 0))
    self.proc()
    self.output()
    # time.sleep(CHAT_WAIT)
    pygame.display.update()
    FPSCLOCK.tick(gc.FPS)
self.quit()
```

HIJACKING THE CHAT SYSTEM...

Stealing user input

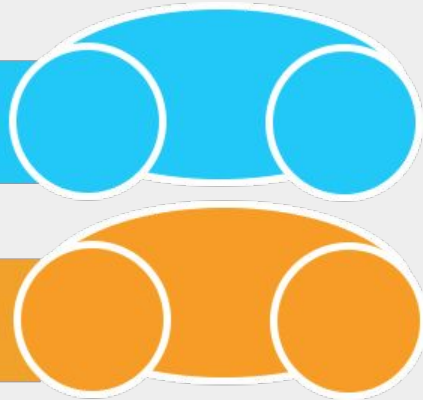
- Part of the S_CHATTING state
- Sends a message to peers when the direction changes

```
pressed = pygame.key.get_pressed()

if pressed[pygame.K_a]:
    if world.players[self.me].changeDirection('left'):
        mysend(self.s, M_DIRECTION + self.me + ":left")
if pressed[pygame.K_d]:
    if world.players[self.me].changeDirection('right'):
        mysend(self.s, M_DIRECTION + self.me + ":right")
if pressed[pygame.K_w]:
    if world.players[self.me].changeDirection('up'):
        mysend(self.s, M_DIRECTION + self.me + ":up")
if pressed[pygame.K_s]:
    if world.players[self.me].changeDirection('down'):
        mysend(self.s, M_DIRECTION + self.me + ":down")

if pressed[pygame.K_RETURN]:
    mysend(self.s, M_START)
```

HIJACKING THE CHAT SYSTEM...

Only need to define three additional message types

```
M_START = 'a'
M_POS = 'b'
M_DIRECTION = 'c'
```

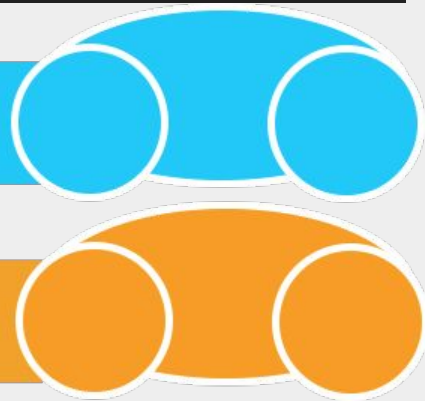


HIJACKING THE CHAT SYSTEM...

Creating a world simulation

- Each client has their simulation
- Update game loop iterations

```
class World():
    def __init__(self, display):
        self.players = {}
        self.trails = []
        self.started = False
        self.display = display
        self.dimensions = (self.display.get_size()[0] // (gc.GRID_SIZE + 1),
                           self.display.get_size()[1] // (gc.GRID_SIZE + 1))
```

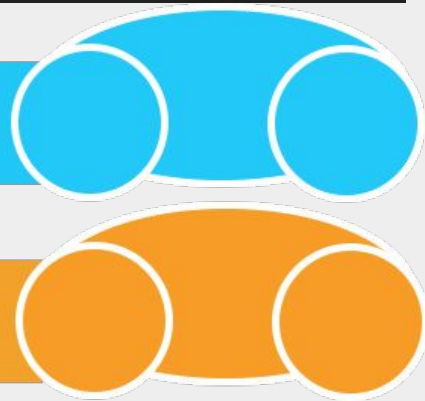


HIJACKING THE CHAT SYSTEM...

Creating a world simulation

- This simulation is persistent

```
class World():
    def __init__(self, display):
        self.players = {}
        self.trails = []
        self.started = False
        self.display = display
        self.dimensions = (self.display.get_size()[0] // (gc.GRID_SIZE + 1),
                           self.display.get_size()[1] // (gc.GRID_SIZE + 1))
```



Pinpoint Trick

1. Pinpoint

```
def pinpoint(msg):
    new_list = list(msg)
    dimension = math.ceil(math.sqrt(len(msg)))**2
    sqrd_dimension = int(math.sqrt(dimension))

    #converting the message to optimal square matrix
    new_list.extend([' ' for i in range(dimension - len(msg))])

    #print(new_list)

    new_list = [
        new_list[j:j + sqrd_dimension]
        for j in range(0, dimension, sqrd_dimension)
    ]

    #row checksum
    row_checksum = []
    for j in range(sqrd_dimension):
        new_row = [ord(new_list[j][k]) for k in range(sqrd_dimension)]
        row_checksum.append(sum(new_row))
    new_list.append(row_checksum)

    #column checksum
    column_checksum = []
    for j in range(sqrd_dimension):
        new_column = [ord(new_list[k][j]) for k in range(sqrd_dimension)]
        column_checksum.append(sum(new_column))
    new_list.append(column_checksum)

    return new_list
```

Pinpoint Trick

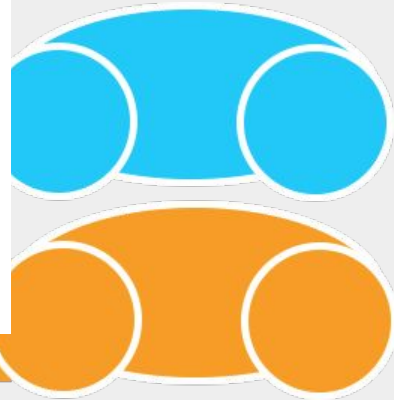
1. Pinpoint
   - Dynamic square matrix

```
In [8]: msg='hi'

In [9]: pinpoint(msg)
Out[9]: [['h', 'i'], [' ', ' '], [209, 64], [136, 137]]

In [10]: msg = "nihao"

In [11]: pinpoint(msg)
Out[11]:
[['n', 'i', 'h'],
 ['a', 'o', ' '],
 [' ', ' ', ' '],
 [319, 240, 96],
 [239, 248, 168]]

In [12]: msg="Yo wassup Fred"

In [13]: pinpoint(msg)
Out[13]:
[['Y', 'o', ' ', 'w'],
 ['a', 's', 's', 'u'],
 ['p', ' ', 'F', 'r'],
 ['e', 'd', ' ', ' '],
 [351, 444, 328, 265],
 [399, 358, 249, 382]]
```

Pinpoint Trick

2. Decode_pinpoint
   - Creating an unstable channel
   - locate the difference(s)

```
def get_point(matrix):
    differences = []
    point_matrix = np.transpose(np.nonzero(matrix))
    points = []
    for i in point_matrix:
        if i[0] == 0:
            points.append([i[1]])
        elif i[0] == 1:
            for j in range(len(points)):
                points[j].append(i[1])
                differences.append(matrix[1].item(i[1]))
    return points, differences
```



Pinpoint Trick

2. Decode_pinpoint
   - Converting matrix back to message

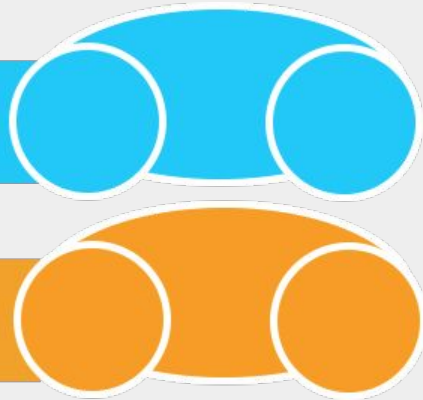
```
def decode_pinpoint(matrix):
    msg = matrix[:-2]
    checksum = matrix[-2:]
    sqrd_dimension = len(matrix[0])
    # print(msg)

    #Locating the error
    new_row_checksum = []
    new_column_checksum = []
    for j in range(sqrd_dimension):
        new_row = [ord(matrix[j][k]) for k in range(sqrd_dimension)]
        new_row_checksum.append(sum(new_row))
        new_column = [ord(matrix[k][j]) for k in range(sqrd_dimension)]
        new_column_checksum.append(sum(new_column))
    new_checksum = [new_row_checksum, new_column_checksum]

    checksum = np.matrix(checksum)
    new_checksum = np.matrix(new_checksum)
    difference_matrix = checksum - new_checksum

    points, differences = get_point(difference_matrix)

    try:
        for i in range(len(points)):
            msg[points[i][0]][points[i][1]] = chr(
                ord(msg[points[i][0]][points[i][1]]) + differences[i])
    except:
        pass

    msg = [''.join(msg[i]) for i in range(len(msg))]
    msg = ''.join(msg)
    return msg.strip()
```

Pinpoint Trick

2. Decode_pinpoint
   - Sending object
   , decoding the object
   received

```
def mysend(s, msg):
    #append size to message and send it
    msg = str(pinpoint(msg))
    msg = ('0' * SIZE_SPEC + str(len(msg)))[-SIZE_SPEC:] + str(msg)
    msg = msg.encode()
    total_sent = 0
    while total_sent < len(msg):
        sent = s.send(msg[total_sent:])
        if sent == 0:
            print('server disconnected')
            break
        total_sent += sent


def myrecv(s):
    #receive size first
    size = ''
    while len(size) < SIZE_SPEC:
        text = s.recv(SIZE_SPEC - len(size)).decode()
        if not text:
            print('disconnected')
            return ('')
        size += text
    size = int(size)
    #now receive message
    msg = ''
    while len(msg) < size:
        text = s.recv(size - len(msg)).decode()
        if text == b'':
            print('disconnected')
            break
        msg += text
    #print ('received '+message)
    return decode_pinpoint(ast.literal_eval(msg))
```

DEMO



FINALLY...

Fork us on Github!:
https://github.com/FrederickGeek8/Reliable-Tron

https://git.io/ICS2017

Things to work on:

- CPU Usage!
- Graphics
- Create separate chat and game rooms

Send us a pull request! Submit issues!