Maze Gaze - A Gaze-Based Multiplayer Game

Kevin Müller
Saarland University
Saarbrücken, Germany
s9kvmuel@stud.uni-saarland.de

Marco Siweris
Saarland University
Saarbrücken, Germany
s8masiwe@stud.uni-saarland.de

Lena Hornberger
Saarland University
Saarbrücken, Germany
s8lehorn@stud.uni-saarland.de

ABSTRACT

This paper is the result of the Seminar "Multi-User Gaze-Based Interaction" at the Saarland University. We designed and implemented an interactive application which uses gaze input. The result of this development is a gaze based multiplayer game called "Maze Gaze" in the category "Multi-User Content Adaption". In this paper we present our project idea, the requirements we made, some design and software decisions and show that a game with gaze input can make a lot of fun despite working with a eyetracker can cause a lot of issues.

INTRODUCTION

Human computer interaction is a discipline of computer science that is concerned with improving existing or coming up with new ways for humans to interact with computer systems. One such interaction style is gaze-based interaction which has it's own challenges. In this paper we examine gaze-based interaction in a multi-user scenario, meaning how multiple users can simultaneously interact with a system using only their eyes. While gaze-based interaction is already well studied, such a multi-user interaction is rather new which gives us the opportunity to gain new insights in this field.

Motivation

Gaze can be a very intuitive and fast way of interacting with a system. However, there are still many unsolved problems related to areas such as tracking accuracy, calibration methods or unintended interactions. Because of that, currently eye-tracking is still very uncommon for end users although applications of it are countless. Researchers are working hard on solving those issues and making eye-tracking more usable in different areas of life, both for personal and professional use.

There is another important fact to consider: People who are partially or fully paralized have a very limited set of possibilities to interact with their environment. Because it is a hand-free interaction, gaze can be a very benefitial for people with physical impairments, giving them many possibilities to interact with other people and perform tasks that otherwise would be very hard or even impossible.

Goal

Our goal in this project is to develop a game that can be played only using the eyes. Evenmore, we want this game to be fun to play and fair against conventional input methods like mouse or keyboard. This way, everyone, including physically impaired people, can have an immersive and competitive gaming experience against other players. By doing so, we hope to gain valuable insights and make a contribution in this field of research.

Outline

This paper consists of three main parts. The first one will lay out the foundation for the project. After examining related work from several perspectives, we will define requirements and make concrete design decisions based on the ideas we got from related papers. In the second part, we will look at the actual implementation of the prototype and explain both hardware, software and how we made it play together. The final part closes the software design cycle by evaluating the game from a performance and user-experience perspective. Based on the results and the lessons we learned along the way, we will draw conclusions and propose future work.

RELATED WORK

Gaze Input Performance

In the year 2009, San Augustin et al. published a paper about their "Evaluation of the potential of gaze input for game interaction." [5] They evaluated two main tasks in game interaction, namely target acquisition and target tracking. The motivation behind their research was to find out whether gaze interaction can, besides yielding higher levels of immersion, increase effectiveness in video games. In their experiment, the researchers used several devices including two eye-trackers, head tracker, mouse, touch screen and a joystick. The target acquisition task asked the participants to move the cursor as quickly as possible two a highlighted point on the screen. Once reached, this point steadily moved back to the middle of the screen, which is called target tracking. From this experiment they gained several important insights. First, giving auditory and movement feedback increases time on target. Second, the target size metters a lot when using an eye-tracker: When the size of the point was sufficiently big enough (150px), the time on target for gaze-tracking was almost as good as for mouse. San Augustin et al. also did a subjective evaluation and found

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

out important factors for the user experience. They found out that tracking accuracy is extremely important. Some users reported that "if tracking is very precise, it feels as if the system was responding to their intentions, rather than to their explicit commands."[5] (TODO check citation) We found this to be very interesting and set to achieve this in our game as well as a goal for this project. As a final remark, the authors claim that the full potential of gaze input in games can only be reached with an accurate and responsive tracker and a well-designed interface.

Mouse Against Gaze Input

We also studied a paper entitled "Gaze Beats Mouse" by Dorr et al. [1]. Although it is from the year 2009 and about 8 years old, we still found it applicable to our problem and especially interesting regarding our own goal for this project. Their main motivation behind this study was the goal to find out if disabled players could play against non-disabled players on an equal level using gaze input. Of course, they were sure that this would not work out of the box but they were eager to find out which gameplay adaptions would be necessary to achieve their goal. They took the popular, yet very simple game called Breakout and adapted it. In the following, we explain some of the adaptions we found to be most related to our game: Gaze-based interfaces in general have problems with confirmation of actions related to the midas touch problem. The authors suggest to make actions automatic whenever possible. Also, giving the user time to practice before the game starts gives them the possibility to better expect and handle the imprecisions. Due to the intuitive interaction of gaze, on can introduce visual complexity to make the game more challenging and fun. They conducted an actual tournament with 20 participants and the results were very clearly in favour of gaze-interaction. The score for gaze was much higher and two thirds of all rounds were won by gaze players. Gaze control was a statistically significant advantage.

Immersive Game Controll Using Gaze

The paper "Gaze-Controlled Gaming: Immersive and Difficult but not Cognitively Overloading" was published in 2014 from Kreytz et al. in Warsaw, Poland [3].
The paper is about controlling games through eye-movements. In the experiment the participants should guide a character trough a maze with their eye movements. It was design within-subjects and had two fixed factors. The first was the game-control type which can be differ in 3 types: gaze-controlled with cues, gaze-controlled without cues and keyboard-controlled. The second factor was the maze complexity(easy and hard). Each participants played all three versions of the game two times, once with the easy and once with the hard maze.
The results important for our project: the completion time was faster with cues, but there were more saccades and there were more saccades with the complex maze, but the participants were gazing on the paths of the maze about 60% of the time.
This resulted the main question: moving the character or scanning paths? For our project this means, that we are switching between two modes: gaze-controlled gaming an visual field scanning. This means we only move the character when the gaze is within a given radius. Moreover we do the game without cues that show possible directions, because this reduce the game experience.

Pursuit Calibration

The paper "Pursuit Calibration: Making Gaze Calibration Less Tedious and More Flexible" was published in 2013 from Pfeuffer et al. [4].
As we know, every application which works with gaze input needs a user calibration before you can interact with it. But a "normal" calibration maybe is difficult and tedious. It requires five or more calibration points. In this paper they present another method to calibrate: "Pursuit Calibration" at which the users are following moving targets to calibrate.
In the experiment they test how the calibration should be: They test different speeds of the moving target and the time the target moves. First the target moves with a constant speed and second with a accelerating moving target which slows down at the corner of the screen and accelerates at the straight side of the screen. The results were: The target does not have the travel across the entire screen, because the final accuracy is already reached around 66% of its whole path and durations that lasts longer than 10 seconds are better.
This resulted how the calibration should be: it should introduce users to the application and be very intuitive (e.g in a stargazing application, following shooting stars). The duration of the calibration should be between 10 to 20 seconds and the size of the target should be, in our case, the width of the path, because the size of the target has an impact on the accuracy of the calibration.
In our project the pursuit calibration was a may have we did not implemented.

REQUIREMENTS AND DESIGN

Gaze Interaction Design Decisions

Various design decisions were very important in our game. Our main theme was the for this project was content adaption. We implement the content adaption by the user having different areas. A User always have a private area and just himself can see this area, if an other player would look to the area, the area is blurred. We realized this by continuously examining the users' gaze on which cell the user is looking. Is this cell, where the user is looking, the cell of another user (that means, he/she was there few seconds before and the cell is lighted in the users' color), so we turn of the light of these cell. If the user looking away, illuminate the light again. But we also have shared areas. That means that two or more player was in the same cell few moments before, and all of them have there a private area. The color of this area is a mix of the colors of the users, where the cell is part of the private area (in this case shared area). So if one of the players if looking of one of the shared cells, it looks like a private area, but simple in another relative color, so the cell is lit. But if a another player is looking of one of these shared cells, and he/she is not part of the commune the cell will be unlit.
The design decisions regarding the colors was very important, because we have to mix the colors for the shared areas. We choose as colors red, yellow, blue and pink. The colors at unity to mix out turned out to be not possible, as we have imagined,

because one got mostly white, since our colors are a light, which is dyed. So then we calculate the mixed colors for the shared areas using the additive color mixing but without dark or lighter. We calculated it with the following Relationships: We have the Domain of a function D a Codomain Y and a Relation $\mathscr{R} \subseteq D \times Y$ with

$$D = \{1,2,4,7\} \quad Y = \{3,5,6,8,9,11\}$$

and the Relation

$$\mathscr{R} = \{(a,b) \in D^2 | a \neq b \wedge c = a + b \; with \; c \in Y\}$$

To assign a color to the numbers we have the following Relation $\mathscr{R}' \subseteq D \cup Y \times C$ with $C = \{red, blue, yellow, pink, purple, orange, violet, green, magenta, cyan\}$

$$\mathscr{R}' = \{(1,red),(2,blue),(3,violet),(4,yellow),(5,orange),(6,green),(7,pink),(8,magenta),(9,purple),(11,cyan)\}$$

We choose the colors in the set C, because these colors are good to recognize and distinguish.
We Also had to decide how large the distance between gaze and light cone should be, so that the player can not control the light cone. At the start we chose a radius of five percent of the screen size. The radius is defined as the center of the circle is the center of the light cone and the end of the circle is the maximum gaze where the user can still control his player. But as we have embedded the tracker, we quickly noticed that it is not a good solution. So we developed an algorithm that calculates where the player most likely looks. We consider only the surrounding cells, and thus reject the control of a radius.

Agile Development Approach

The main advantage of taking an iterative approach in the development of such projects is, that one can get working prototypes really quickly and adapt the goals and requirements based on the evaluation of those prototypes. Also, by doing vertical prototypes, i.e. ones that only implement a single unit of functionality, technical limitations can be uncovered early in the development.
Initially, in our small team of three, we defined high-level milestones with deadlines that would eventually lead to the completion of the project within the given time frame. Each week, we met to discuss and assign smaller action items called ”issues” to each team member using GitLab. Our initial focus was on developing two prototypes that would ensure our idea was possible. We implemented a horizontal prototype with a limited set of features, very basic graphics and only mouse and keyboard controls. By testing this prototype with some users, we could quickly find out that our game is indeed fun to play and the competitive nature of the game does work.
Because this has not been done before and there was no reference implementation online, we did a vertical prototype covering the eye-tracking functionality to ensure we could receive the gaze data from multiple eye trackers. After a while, this worked and we could focus on putting both prototypes together into a working gaze-controlled game and increasing its technical and visual fidelity.

Formal Requirements

Must-Haves

- You could play the game on a conventional Laptop display or TV set
- It could play 1-4 players at a time
- Player can enter a running game
- The match field is a maze consisting of walls and paths
- Maze is random generated. The Maze does not change in playtime
- Each Player starts in one corner of the Maze
- At the beginning of the game is the hole maze dark
- The play figure of a player is a colored light cone
- Each player has a different color
- Light cone follows the gaze of the player
- Light cone tries to follow the gaze in a constant speed and tries to minimize the distance to the gaze
- If the distance of the gaze an the light cone is too big, the gaze can no longer be control the light cone.
- If the light cone can not minimize the distance to the gaze because of walls, he stops
- A small part around the light cone is lit
- The paths on which the light cone had previously been left also remain bright, but gradually darken
- illuminated area is only for the player visible how reached the are
- If a player’s gaze moves near a path discovered by another opponent and not by himself, it is completely darkened
- The light cone of the player is always lit
- Lightened areas a player himself has discovered will not be darkened for this player, even if they were discovered by another player
- At the beginning of a round, a target is placed at a random position in the maze
- It gave different good power ups (enlightenment, show tart, endurance)

May-Haves

- Player can choose different levels at the beginning of a round, thereby increasing the size of the maze
- New players could join a game, without pausing
- More different bad power ups (dim, slowing, darkness)
- If a player does not look at the field for more than 10 seconds, he leaves the game and his light cone disappears from the field

IMPLEMENTATION

Hardware

We used the pupil framework developed by pupil labs [2] for this project. Pupil is an open source platform for mobile eye tracking and can be used for a variety of mobile eye-tracking applications. Although the software has its bugs as we had to find out, it is still a lot more sophisticated than it was some years ago and tracking performance is very good. The headset consists of two cameras. One front facing webcam is capturing what is in the view of the user. A second smaller camera is located slightly below one eye and captures the whole eye. The software extracts the pupil from this image of the eye. The framework comes with a set of calibration methods that provide a way to map eye positions to points on the world view camera image. A user of the platform can easily add new plug-ins such as surface tracking or network capabilities to the basic pupil capture application. Some issues we discovered with the software and hardware were that the eye camera quickly gets very hot and needs to cool down. Also, the cameras are quite frequently not recognized by windows and drivers need to be reinstalled. The calibration sometimes crashes the application while other times theuser's pupil can't be recognized reliably.

Software

The main crux in this project was developing a software system that integrates well with the given hardware and is fun to play. Also, there should be some experimental aspects to the implementation. In the following, we will show how we achieved this by explaining the key components of the software. The whole application was built using the Unity game engine with the programming language C#. Implementation and testing took place on regular windows machines. We used git as VCS and GitLab to host our code and track issues.

Menu

At the beginning of the game the user can choose between many game configurations. There are the options to start the game, to finish the application, to get some help information (e.g. link to GitLab) and to call settings, where the user can connect new clients to the application, disconnect connected clients and calibrate clients. On starting the game the game configurations can be set. The user can choose the number of players, the number of rounds the user wants to play and the difficulty of the game (simple, medium or hard) which affects the size of the labyrinth and the paths. If the game lasts longer than one round the next round starts automatically after a short break. The game also supports in-game interruptions with a break button. In this interruption mode the game can be continued and also aborted by going back to the main menu. All these options can controlled with the mouse.

Maze Generator

The Maze a generated with a depth-first-search algorithm. First we generate a $n * m$ large matrix and in every step we have a wall from x_0 to x_n and from y_0 to y_m. In the next step we generate a datatype Cell with four walls per Cell (north, south, east, west). Now we will chose a random start cell, and destroy a random wall which has not yet been destroyed from this cell and progress with death-first-search in every cell of the matrix. You can solve this problem with a running time of $\mathcal{O}(n^2)$.

Power-Ups

We distinguish the power-ups in good and bad power ups. The good power-ups are enlightenment, endurance and show target. Enlightenment is a light explosion, if a player collect this target, a big range of nMazeSize$/7*$ mMazeSize$/7$ will lit. Endurance is a power-up which bring about light path is lit for a longer time. The power-up show target trigger a light flash on the target for one seconds. The bad power-ups are dim, slowing and darkness. Dim is the opposite of endurance, so the the light path is lit for a shorter time, but only of the opponents. The power ups slowing produce that the opponents getting slower and the slower speed is fix, and the player can not get his old faster speed. Darkness extinguishes the light of the opponents.

Smart Player Controls

We quickly discovered that simply moving the player to the position of the mouse of gaze does not work very well because the player has to keep the cursor very close to his light cone all the time. Also, this would not look very good because the lights will constantly move along the walls. We therefore decided to implement smart player controls, which means that the light will move towards the position of the cursor, but not in a straight line but through the corridors of the maze. For example, if the cursor is around a corner, the light will move first horizontally and then vertically around the corner, but never diagonally directly towards the cursor.

We noticed another problem with gaze: Oftentimes, the gaze is not very accurate and jumps around quite a lot. When we receive the gaze data from the eye tracker we convert this information into a cell in the maze. However, instead of blindly trying to move the light to this cell, we also look at the neighboring cells. For each of them, we compute a simple heuristic which says how likely it is that this is the intended cell that the user wants to move to. So, even if the gaze position jumps around the intended cell, our algorithm will always give the cell that makes the most sense to move towards. This was not trivial to implement and we did not know if it would actually work. However, when we tested this approach, we found that moving the light through the maze is a lot more efficient. Without this approach, oftentimes the light went back and forth or stood still for prolonged periods of time. Now, we can move the light through the maze almost without errors or pauses.

Currently, this simple heuristic only takes into consideration whether the actual gaze position is within reach for the player and the distance of the path to reach it. Even though this is so simple, the results have been quite impressive. Improving this heuristic will probably lead to even better results.

Game Fairness

We already mentioned the length of the path between two positions in the maze in the paragraph above. Obviously, getting this path is not straight-forward and we first felt that we need this functionality when we wanted to spawn our target fair, such that all players have the same chance of reaching it. In the first prototype, we used the simple Manhattan distance to

find the position with roughly the same distance to all players. But of course, depending on the structure of the maze, this can lead to quite unfair target positions. We therefore implemented a pathfinding algorithm that takes two cells as input and returns the path (i.e. list of cells to visit) to get from the start cell to the end cell. By utilizing this, we can compute the spot in the maze which each player can reach in roughly about the same time when making perfect decisions in choosing the way. We also introduce a lower bound on the length of the path to ensure that the target will not always spawn right between two players. When a new round starts, we generate new mazes until we find one where we can spawn the perfect target. When playing multiple rounds, we gradually make the spawning algorithm more liberal such that it will eventually spawn the target somewhere because we can't generate a new maze in between rounds.

Pupil Integration

We created a class that stores the client information like ip addresses, ports and surface names. After those configurations have been read from the saved settings, we hand them over to the pupil listener. After validating the provided IP address and port, we establish a request socket and send a request for the sub port to the instance of pupil capture.

```
requestSocket = new RequestSocket(
  IPHeader + c.port);
requestSocket.SendFrame("SUB_PORT");).
```

After receiving this subport, we go on to set up the subscriber socket and subscribe to surface event of pupil capture.

```
 SubscriberSocket subscriberSocket =
   new SubscriberSocket(header +  subport);
subscriberSocket.Subscribe("surface");
subscriberSocket.Subscribe("notify.")
```

We then enter a loop that continuously tries to read data from that socket.

```
 bool stillAlive = subscriberSockets[turn].
  TryReceiveMultipartMessage(
    timeout, ref (msg));
```

Pupil provides this data as soon as it detected a defined surface within the view of the user. The message is a json file containing all sorts of information about the gaze and the surface. We extract the relevant information, which is whether the gaze is on the surface and the relative positions on the surface ranging from 0 to 1. This data is passed on to the gaze controller which converts this data into actual screen corrdinates which are then used to determine where to move the light. All of this is done in turn for all of the four players because due to a restriction in unity we can't connect to all four pupil instances at once.
Calibration is done in a similar fashion. When the calibration needs to be started, we send a request to the pupil client

```
sendRequest(requestSocket,
   new Dictionary<string, object> {
     { "subject", "calibration.should_start" },
     { "marker_size", "1.50" },
     { "sample_duration", "50" }
   }
);
```

This starts the internal pupil screen marker calibration. We can then find out when the calibration is over by checking for each received message whether the message type equals a certain event id).

```
if (msgType.Equals(
  "notify.calibration.successful")) {
// end calibration }
```

Calibration

In order to use the pupil trackers and have a good level of accuracy, a calibration that maps eye positions to locations in the view of the world camera is necessary. When a calibration is done, the tracker must not move on the players head. Otherwise, the slight shift will introduce a more or less severe offset which quickly leads to very bad usability. To account for that, we introduced a calibration procedure in our application. We decided to do this in a rather simple way: From the settings screen where we can connect to pupil clients, we can also choose to start the calibration procedure for each client. This will send a request to the client computer and start the pupil internal screen marker calibration procedure. Our game then listens for an event from pupil, indicating a successful or failed procedure. Furthermore, when a new game starts each player that has not been calibrated previously will be calibrated in the same way. When the calibration for every player is done, the players can join the game.

Game Lobbys

We implemented a very intuitive and efficient way for gaze players to join a new game. Because our maximum number of players is four, we divided the screen into four areas. When a player gazes at one of those areas, a countdown from 3 to 0 is shown to indicate how long they have to keep looking there to join. After all players have joined in that manner, the game will start automatically. To ensure that games don't get empty, we implemented the possibility for mid-game joining. When players are inactive, i.e. they do not gaze at the screen for a certain number of seconds, they will be kicked out of the game. New players can then join the ongoing game in the same manner as described above, with the only difference that the other players can keep on playing. The new player will then join the game and spawn when the next round starts.

Graphical Interface

To get a immersive game experience we integrate special effects and high quality 3D objects. The special effects come in many places in the project. We took our effects from "Particle Ingredient Pack" it is a particle System package from the asset store from unity. In this package are located so many example particle effects, we choose a two SFX for the power-ups. One for the good power-ups and one for the bad power-ups. We change some parameters of these two effects (i.e. remove the loop, change the color, etc.) and trigger them, if a user collect a power-up. To make the game more interesting, we integrate a fog. The fog spawned irregularly and accidentally and only from the second round. It makes the control more difficult.
To display the power-ups, we took a Gem package called "Gem Shader" from the Asset store from unity. We choose

3 different Gem to distinguish the target, a bad and a good power-up. We use a ingredient from the "Particle Ingredient Pack" called "Ingredient-Field(Circle)-Flash" to represent the players.

EVALUATION

Performance

We tested and played the game a lot of times and it turned out that the performance of the game played with gaze input is very good. Depending on the hardware used to play the game, our game software is running without any problems. The only problem we were faced with, the eyetracker was not always accurate, but we avoid this problem, by not showing the exact position of the gaze and implemented an method, that estimate the best gaze position of the user. With this method the performance of the gaze input increased a lot. Now the game can be played with gaze, keyboard and mouse input, and everybody has the same chance to win.

User-Experience

We interviewed some people, most of them haven't played a game with gaze input till now. Because of that, they needed little time to get used to the eyetracker and the game. They said, the game and gaze control were very intuitive, easy to learn and easy to remember. They also liked the design of the game: The background of the main menu, that it has the same topic and the reference to actual game: maze, the slow music first in the menu and the faster music in the game which increases during the rounds and creates a lot of pressure and also the color of the individual pawns in the game (the cone of light) which makes which makes it easier for the observers to differ the players. A couple of people also make new suggestions to improve the game and make it more fun, but we had not the time to implement all these ideas.
Summarized all interviewed people have the opinion that the game is interesting, because they haven't used eyetracker and gaze input before, but it they had all much fun playing our game.

CONCLUSION

Résume

We were faced with a lot of problems during the development, especially with the eyetracker. But we found ways to solve or avoid this problems and created a game that can be played only with gaze input.

Lessons Learned

The main benefit of having done this project is that we have learned lot's of important lessons. Besides lessons in project management and software engineering practices, we will explain the key insights related to gaze interaction. While some of them were already mentioned in related work, we also gained some new ideas from our unique approach and experimentation with different settings and implementations. We would like to present those findings as guidelines for others to implement in their eye-tracking projects.

- Make hit areas bigger than visual parts.
- Use more than four markers and make them as small as possible and as big as necessary to increase stability and precision
- Accurate tracking is important and offsets are frustrating. Do not show the actual gaze position as cursor whenever possible.
- Make actions automatic whenever possible.
- Use two-modes for player control and scanning. This can be achieved by defining a fixed radius around the player where movement is possible.
- Estimate the best user gaze position. Do not take the raw gaze data but find a heuristic that yields a better estimate of what the user intends to do in the given state of the game.

Future Work

In general, there are many possibilities to extend the game MazeGaze. In stead of control the menu by mouse we could implement a gaze bases control menu. So for example the user have to look in different areas to choose different options in the menu. For instance the person has to look in the upper left corner to choose the game option to play in a small maze. Another approach would be, the user has to control a light cone in certain corner to choose something. This is also a good way of training before the the game starts.
We also could create a collaborative team mode so that two players are in a team and other to players are in a opponent team. The teams have to play against each other. That means, if a player of the one team collect a bad power-up it just will be triggered on the members of the other team and not for the player of the own team.
Another idea would be to change the maze in play time. Like the game "Labyrinth" from Ravensburger in 1986. A algorithm destroy walls and build up other walls, but still it is a fair maze, so every player can reach each point of the maze. This ensures that the users can not only rely on their memory, but also a luck factor plays a role.
Another point is the continuation and improvement of the calibration. We could implement a In-application-calibration, so that means if we press the calibration button, the calibration will start on the desktop where the game is running and not on the desktop where the eye tracker is connected. This ensures a better and more precise calibration. Also we could implement a in-application pursuit calibration to get a more precise calibration and better results and a low frustration of the user.
To reduce the frustration of the user, we could improve the gaze precision. The gaze precision is currently good enough to have a good gaming experience without major frustrations but not perfect yet. The improvement is possible by adding algorithms to the actual gaze position calculated to the desired position.
Last but not least we could implement a queue to start waiting players in a fair, right sequence. If, for example, 5 people want to play, we have a problem, because the game is design for four users. So the fifth user have wait and getting in a virtual queue, when one of the four players leave to game, the waiting person will spawn in the next round. If we have two or more

waiting users, they are arranged in a queue, with the principle, first come first serve.

TOTAL: 8 PAGES

REFERENCES

1. Michael Dorr, Laura Pomarjanschi, and Erhardt Barth. 2009. Gaze beats mouse: A case study on a gaze-controlled breakout. *PsychNology Journal* 7, 2 (2009).
2. Moritz Kassner, William Patera, and Andreas Bulling. 2014. Pupil: an open source platform for pervasive eye tracking and mobile gaze-based interaction. In *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: Adjunct publication*. ACM, 1151–1160.
3. Krzysztof Krejtz, Cezary Biele, Dominik Chrzastowski, Agata Kopacz, Anna Niedzielska, Piotr Toczyski, and Andrew Duchowski. 2014. Gaze-controlled gaming: immersive and difficult but not cognitively overloading. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, 1123–1129.
4. Ken Pfeuffer, Melodie Vidal, Jayson Turner, Andreas Bulling, and Hans Gellersen. 2013. Pursuit calibration: Making gaze calibration less tedious and more flexible. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 261–270.
5. Javier San Agustin, Julio C Mateo, John Paulin Hansen, and Arantxa Villanueva. 2009. Evaluation of the potential of gaze input for game interaction. *PsychNology Journal* 7, 2 (2009), 213–236.