

Understanding Playtest Data through Visual Data Mining in Interactive 3D Environments

Priyesh N. Dixit and G. Michael Youngblood
The University of North Carolina at Charlotte
College of Computing and Informatics, Department of Computer Science
9201 University City Blvd, Charlotte, NC 28223-0001
{pndixit, youngbld}@uncc.edu

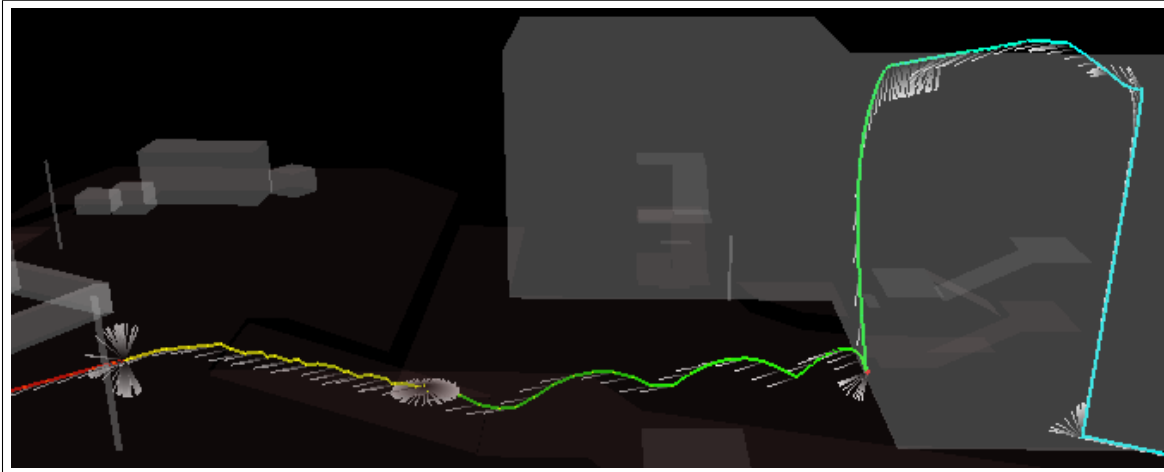


Figure 1: A player trace showing four of the player behavior patterns we discovered using the visual data mining process. From right to left: emergent behavior, jumping, pirouette, and fluster.

Abstract

Creating an interactive 3D environment can be challenging, in particular because every user (player) will interact with the environment in their own unique manner. Gaining feedback through playtesting is a common method for evaluating an interactive experience. However, the information gathered from playtesting is often subjective and hard to interpret. Visual representation of logged game player data can provide insight in order to discover patterns and behavior that would otherwise be hidden. In this paper, we explored the use of visual data mining and automated data processing to process the logged player data and present the results of finding five unique phenomena in our dataset of 3079 players. These phenomena include pirouettes, flusters, jumpers, learning, and emergent behavior. Using the visual data mining process, we can find occurrences of such behavioral patterns in order to understand player behavior and improve the interactive experience.

CR Categories: I.6.3 [Simulation and Modeling]: Applications; I.6.6 [Simulation and Modeling]: Simulation Output Analysis;

Keywords: visualization, interactive data mining, game environments

1 INTRODUCTION

A common method used for creating interactive environments is subjective design where the designer creates the game experience assuming the players will interact with the game in a certain way [Hong 2005]. This subjective evaluation is not always sufficient since the behaviors of a human player in a game environment can vary greatly [Tracy Fullerton and Hoffman 2004]. In addition, a player who has never interacted with a game before will behave very differently from an average or even an expert player. In order to gain insight into how an average player might interact with the environment, we must get several players of varying experience levels to play the game before it is finished. This process of allowing players to play a game pre-release is known as *playtesting* [Tracy Fullerton and Hoffman 2004]. Some companies, such as Valve Software [Brashill and Barnett 2007], have integrated playtesting heavily into their development process, which has enabled them to better understand player interaction. However, the evaluation of this playtesting is often done in a subjective manner through interviews or direct observation [Niedenthal 2007]. We propose the use of tools that allow for objective evaluation of the playtesting by collecting data about the playtesters' interactions with the environment.

In order to use tools to evaluate playtesting we must first record data from the player's play session that gives us the information we need to evaluate their behavior in the environment. There are several variables we can track for a player in a game environment such as position, orientation, score, health, when they fire a weapon, and so forth. However, the variables that we focus on are their position and orientation recorded at a certain frequency for the entirety of their play session. These variables represent the most prevalent and

consistently changing components of the normally collected data in first and third person games—our targeted area. The position is the spatial coordinate of where they are in the environment (e.g., x,y,z). The player’s orientation consists of their yaw, pitch, and roll angles as they explore the environment. However, once this data is recorded, the challenge is how to interpret it. The data can be overwhelming and hard to decipher. For instance, we record the player’s position and orientation information (6 variables total) at 10 Hertz, which results in 60 data points per second. This means that for one hour of game play we collect 216,000 data points. In order to make the data easier to understand we can represent it visually. The goal of visual representation of the data is to allow the human to draw conclusions and gain insight into the data more quickly. This is known as *visual data mining* and is especially useful when it is not clear what is being searched for and little is known about the data [Keim 2002].

By processing the player’s position and orientation information, we can derive knowledge about their behavior that can help guide the design process. In order to process this information, specialized tools are needed that will read the player data and present the visual representation in an interactive manner. We have developed a tool, known as *PlayerViz*, as part of the Common Games Understanding and Learning (CGUL) toolkit for processing the player data.

PlayerViz allows the user to interactively visualize the player data. In order to process a large number of players, we used *PlayerViz* to visualize multiple positions of the world view for each player trace in our dataset. We then organize these images into a collection of web pages for easy viewing and sharing by an analyst. Using human-guided visual data exploration with these web pages, we were able to find interesting patterns of behavior to investigate further. Once we know what we are searching for, we can develop mathematical models for the patterns and search through the dataset for other occurrences. Using this model, we have found several patterns for player behavior that can be discovered (i.e., “mined”) from the data. Once found, knowledge can be derived from these patterns of behavior, which can aid in the design process of the interactive environment.

2 Problem Statement

The key question we are trying to answer is: *How do we derive knowledge about player behavior through visual data mining in order to better guide the design of an interactive experience for a wide range of players.* Usually, playtesting consists of observation of the players by the design team, oral interviews and feedback forms [Niedenthal 2007]. Logging of player data can also be used to gain feedback from the playtest sessions because it can make the process more efficient. The development team could capture the tester’s session in video form but it is time consuming to watch the videos, videos consume a lot of disk space, and automated video processing is a difficult task. Another problem with video or screen capture playback is that it is often difficult to get a complete picture of play just from the player’s perspective. Watching the session first-hand can introduce bias from the observer’s opinion. Seeing the entire path or desired sections of the player’s interaction in an interactive analysis tool such as *PlayerViz* would be very useful in understanding the behavior of that player.

The most common reason to exclude or oversimplify logging is that it can slow down the game. For instance, during the development for *Age of Empires II: Age of Kings* the developers used faster machines for playtesting than targeted deployment due to the slow down of logging playtest data [Marselas 2000]. This does not always have to be the case because one of the most machine expensive steps in logging is file IO, which can be done more op-

portunistically at periods of lessened hardware need, at cut-scenes or designed lulls, or by using multiple threads instead of constantly during game play. There is also an issue of the needed fidelity for logging. Plenty of information can be gained from 1 Hz or slower logging, so 10 Hz or better is not always necessary. Speeds higher than 10 Hz are especially not needed due to limitations in human reaction speed [Boff and Lincoln 1988].

We will first discuss some related work in both visualization and game play analysis, then discuss our methodology for the analysis of player information and provide details on the implementation of the analysis tool. Once we discuss the implementation, we detail our experimentation, and the results. Finally, we outline our observations from the results of the experiment and suggest some future work that can be done to further this research.

3 RELATED WORK

Visualization is the use of computer-supported, interactive, and dynamic visual representations of data to gain new insights and form new hypotheses [Meyer and Cook 2000; Keim 2002]. Visual data exploration is invaluable when the dataset is unfamiliar and it is unclear what patterns will be found. It is important to include the human in the data exploration process in order to combine the creativity and domain knowledge of the human with the computational power and storage capacity of computers [Keim 2002]. There are several techniques for the visualization of large datasets. The method we use to visualize the player is similar to the iconographic technique in which a glyph is used to represent a data point with its various features being influenced by the data [de Oliveira and Levkowitz 2003; Keim 2002]. In our case, the glyphs are the connected spheres that represent the player’s logged information at that particular time step.

There are many examples of visual data mining, but from recent surveys of the field the most related to our work is the visualization of web server logs for knowledge discovery [de Oliveira and Levkowitz 2003]. The paths of users navigating the web site are visualized in an interactive manner. An analyst can then study the visual data to find interesting behavioral patterns in order to improve the end user experience. These types of behavioral patterns mainly focused on navigation patterns in webpages (i.e., information flow). Another similar application of visual data exploration of web usage was also conducted through an application called *WebViz* [Jiyang Chen and Goebel 2007]. They collected a large amount of user data, visualized it using an interactive tool, and then discovered patterns in the data through analysis. These patterns included connectivity across sites (i.e., information structure). We use similar methods to these examples, but instead apply it to 3D game player data.

Visualization of logged player information in a game environment has been examined in the context of massively multiplayer online games [Joslin et al. 2007]. Their approach was to integrate the logging and visualization tools into a specialized build of the game. The *game logging* build also allowed for additional user interface elements for the playtesters to report game play issues (e.g., if the section they just played was fun or not). Integrating the analysis tools into the game has several advantages such as making the tools easier to access and the ability to quickly analyze data that has been collected during a play session when interesting patterns are noticed. However, this approach is limited in that it is very application specific. Our approach is different in that we focus on first collecting the data and then processing it for analysis after the playtesting is conducted. Our tools are also designed across the first and third-person game genres and not necessarily specific games.

A tool similar to *PlayerViz* called *VU-Flow* was developed for

tracking user behavior in a virtual environment [Chittaro et al. 2006]. The user’s position and orientation information is captured, but it is visualized only in two dimensions. The tool was used to track areas most visited (and least visited) as well as parts of the environment that were most seen (or least seen) by the users. Our approach for visualizing tracked player data is in full 3D and is capable of visually representing more information about the player’s behavior such as shots fired and loss of health.

Bungie Studios recently revealed their extensive testing lab, which is used not only for finding glitches and bugs, but also to improve gameplay through playtest data [Thompson 2007]. In one example, the players would often become lost and wander for long periods of time in one of the levels, which was problematic. In order to resolve this issue, they changed the level layout to be more linear and explicitly guided the player down the right path. They use what is called a *heat map* (such as the ones in Figure 8) which visualizes where the players spend the most time by color. If many players visited a particular point on the map it would be colored red, if few players visited the area it would be colored blue. They also store video of each player’s play session for later review by the testing team. The use of a visual data mining process would allow them to extract knowledge from a very large collection of players. This would be useful if the large number of online players of a game like *Halo 3* were logged and analyzed.

Valve Software also conducts extensive playtesting of their games, according to the in-game commentary in their first-person puzzle game *Portal* [Brashill and Barnett 2007]. One instance where the design of the game was altered through playtesting was a level where the player was required to look up to find the exit. However, most players rarely looked up. In this case, they added a broken ladder to prompt the player to investigate where it leads and thus look upward. The use of a visual data mining process conducted using recorded player traces would have shown most players getting stuck for long periods of time. This knowledge would allow the designers to place a hint at a highly visited location for the player. The methodology used for conducting the playtesting in *Portal* is unknown. However, Valve Software does log player information and has posted some of the results online [Software 2007]. The statistics that they track include average play time, number of deaths, and also a *heat map* showing the areas where the most deaths occur in individual levels.

The use of visualization for knowledge discovery in a dataset of logged user information has been explored by others. However, there are few applications of this method to game player interaction information. The visual analysis of users in a game environment conducted by others has generally been specific to a certain game or constrained by 2D. The use of our tools is generalized to any first or third-person interactive experience that can output the position and orientation of the player over time.

4 METHODOLOGY

At the core of our methodology is the process we used for visually representing the player’s logged actions. This representation is called a *player trace* and is shown in Figure 2. The minimum information needed to render a player trace is the player’s spatial position (in 2 or 3 dimensional space) and their orientation (the yaw, pitch, and roll angles). Each of the player’s positions over time are marked by a small sphere centered at the point where the player was located and the player’s orientation at that time is represented by an oriented line segment protruding from the sphere. The spheres are then connected by line segments to represent a path where each sphere is a node in the player’s path. Color cycling is used to represent the flow of time using a blue-green-red rain-

bow color scale. The color scale is normalized across the player’s path such that bunching of colors indicates that the player is moving slower and the same color stretched across multiple spheres indicates faster movement. Since the color scale is normalized, the path always starts with blue and ends in red. It is assumed that this data is logged at a consistent rate throughout the play session. In our implementation, we logged the player’s information at a rate of ten times a second (10 Hertz). However, a lower frequency can still provide useful information about the player’s behavior (a situation that may be necessary to facilitate logging with processor intensive simulations or games).

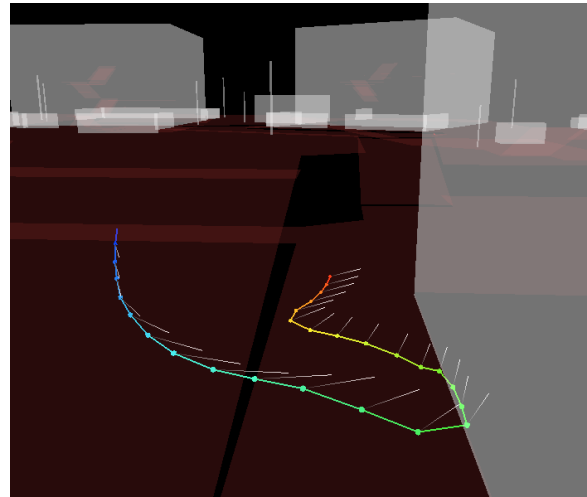


Figure 2: Screenshot from *PlayerViz* of a player trace and the positive (gray) and negative (red) space regions.

Seeing the player trace alone does not always provide enough information in order to interpret it. We must also know the environmental context under which the player trace was recorded (i.e., the geometry of the environment). However, the geometry does not need to be shown in great detail and can be greatly simplified. There are two types of world space to consider, positive and negative space. The positive space is the solid world geometry that the player can see such as buildings and lamp posts, which are created by the level designer as simplified bounding regions for the large geometric objects in the world¹. The negative space is the empty space between positive regions (and sometimes within a positive space region) that the player can navigate. The negative space regions can be manually defined by the level designer or automatically generated using a technique such as the Hertel-Mehlhorn decomposition algorithm [Hertel and Mehlhorn 1983].

Once we have the positive and negative space regions, we can represent them as 3D polyhedrons alongside the player trace to show the player locations relative to the geometry. The positive space regions are rendered in gray with transparency. The negative space regions are reduced to the bottom most polygon and rendered in red as “floor” polygons. The rendered regions can be seen in Figure 2.

4.1 Deriving behavioral patterns

Once we load a player trace and the world geometry into the *PlayerViz* tool, it is usually quite clear what happened while the player was interacting with this environment. By processing this visual information and applying domain knowledge, we can further under-

¹Most level design tools can automatically generate the simplified geometry from the actual models as bounding boxes.

stand the behavior of the player. However, if the number of player traces recorded is very large, it is not reasonable to load each one into the tool manually and view them one by one. In order to allow a human to quickly skim through a large collection of player traces, it must be presented in an organized fashion. We developed a program that generates a collection of web pages showing every player trace captured in a particular level from four camera angles—the corners of the world in the skybox taken at a high vantage point looking towards the center of the ground-plane. This allows a domain expert to visually explore the dataset of players and search for interesting patterns of behavior that should be investigated further. Since we did not know the patterns for which we were searching, we could not automate the search. However, once the pattern was discovered we developed a math model by which to automatically detect its presence. Once this model was developed, we could search the rest of the dataset and find all other instances of this behavior. We used this process on our dataset of 3079 player traces and the resulting patterns found are outlined in Figure 3. Once we discovered these phenomena, we wrote a program to automatically process the player traces and find instances of these phenomena using Algorithm 1. This algorithm can be run in real-time as the player is playing, in which case one could respond to the patterns as soon as they appear.

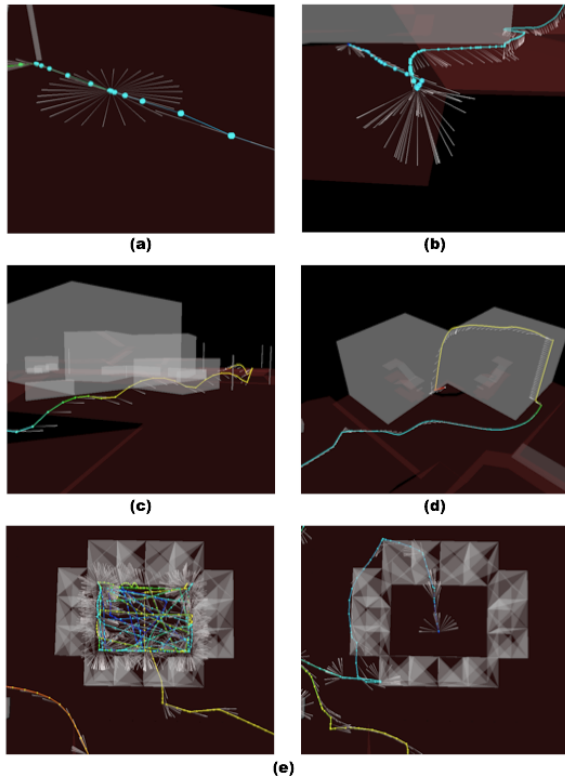


Figure 3: Player behavior patterns found through Visual Data Mining: (a) pirouette (b) fluster (c) jumper (d) emergent behavior (e) learning

While visually analyzing the data, we noticed that players sometimes turn nearly around 360 degrees while standing still. We call this a *pirouette* and an example is shown in Figure 3(a). This pattern usually indicates that the player is lost or is reorienting themselves and often happens at the start of a play session when the player is unfamiliar with the environment. This can also occur if the player encounters an area of the level with an ambiguous design and they

Algorithm 1 Processing a list of player traces

```

for all playerTrace in playerTraceList do
  Clear patternList
  previousNode = null
  // Patterns involving matches across closely clustered positions
  for all traceNode in playerTrace do
    lastDistance = distance(previousNode,traceNode)
    if lastDistance < THRESHOLD then
      Add traceNode to patternList
    else
      if patternList is not empty then
        // We now have a list of nodes that were close together
        // this indicates the player was not moving much
        if isFluster(patternList) then
          Found fluster
        end if
        if isPirouette(patternList) then
          Found pirouette
        end if
        Clear patternList
      end if
      if isJumper(playerTrace) then
        Found jumper
      end if
    end if
    previousNode = traceNode
  end for
  // Patterns involving matches across an entire trace
  if isJumper(playerTrace) then
    Found jumper
  end if
end for

```

lose track of which direction they need to go. Algorithm 2 shows the algorithm for calculating a pirouette from a given list of player trace nodes. If we detect that the player has performed a pirouette, we could assist them by turning on a navigation arrow or highlighting the direction in which they need to go. Understanding this behavioral occurrence can also guide a level designer to add unique features to help orient players in areas where pirouettes occur.

Another interesting pattern we discovered was when the player's look direction gets stuck looking in a relatively confined cone with rapid changes in view for a period of time such that the visual image looks like a sphere with a multitude of protruding line segments. We call this phenomenon a *fluster* and can be seen in Figure 3(b). It is usually caused by a loss of mouse control or high mouse sensitivity. A fluster can also occur due to gimbal lock in the player's rotation angles. Gimbal lock occurs when two of the three rotation axes become parallel due to the rotations made by the player. This results in the player being locked in one look direction for a short period of time. Algorithm 3 explains how to detect a fluster given a list of player trace nodes. If a fluster is detected as a player is playing the game, we can force their look direction to a certain angle to help them escape the fluster. It can also be used to detect mouse driver failures or differences between controllers that cause such behaviors.

Using the visual representation of the player's activity we were able to find instances where the player constantly jumped through the environment. We call these players *jumpers* and an example can be seen in Figure 3(c). Constant jumping is a strategy that some advanced players use to evade attack and to move faster through the world. Jumpers tend to see less information because they are moving faster than non-jumpers, and they also tend to be more experienced players. The algorithm for detecting if a player is a jumper is specified in Algorithm 4. If we detect jumpers as they are playing, we can highlight important information artifacts more prominently to compensate for their increased speed, thus providing a more con-

Algorithm 2 Calculating a pirouette

```
define isPirouette(NodeList patternList):
    previousYaw = 0
    // Process change in horizontal angle (yaw)
    // yaw range is from -180 to 180
    for all patternNode in patternList do
        currentYaw = patternNode.yaw
        //Shift yaw range to [0,360] instead of [-180,180]
        if currentYaw < 0 then
            currentYaw = 360 - abs(currentYaw)
        end if
        if previousYaw < 0 then
            previousYaw = 360 - abs(previousYaw)
        end if
        //Calculate difference in yaw
        deltaYaw = currentYaw - previousYaw
        if abs(deltaYaw) > 180 then
            //yaw crossed the zero angle
            if deltaYaw > 0 then
                deltaYaw = -(360 - abs(deltaYaw))
            else
                deltaYaw = 360 - abs(deltaYaw)
            end if
        end if
        totalYaw += deltaYaw
        previousYaw = currentYaw
    end for
    //Check if they turned more than a certain angle
    //(e.g., 300 degrees)
    if abs(totalYaw) > MinimumAngle then
        return true
    end if
    return false
```

Algorithm 3 Calculating a fluster

```
define isFluster(NodeList patternList):
    previousPitch = 0
    for all patternNode in patternList do
        // Pitch range expected to be [0,+180]
        totalPitch += abs(patternNode.pitch - previousPitch)
        previousPitch = patternNode.pitch
    end for
    // Pitch entropy is the average change in pitch for this pattern list
    pitchEntropy = totalPitch / patternList.size
    // Check if this average change is greater than the given threshold
    //(e.g., 7 degrees)
    if pitchEntropy > FlusterPitchEntropy then
        return true
    end if
    return false
```

sistent experience across different player types.

Algorithm 4 Calculating if a player is a jumper

```
define isJumper(NodeList playerTrace):
    // We assume here that the up axis is +Z
    previousZ = 0
    for all traceNode in playerTrace do
        totalZ += abs(traceNode.z - previousZ)
        previousZ = traceNode.z
    end for
    // zEntropy is the average change in height for the player
    zEntropy = totalZ / playerTrace.size
    // Check if the zEntropy is greater than the threshold
    //(e.g., 4 world units)
    if zEntropy > JumperEntropy then
        return true
    end if
    return false
```

Sometimes we also find that the player acted in ways that we did not anticipate. This is known as *emergent behavior* and an example is shown in Figure 3(d). In this example, the player was expected to climb over the small wall and reach the goal. Instead this player climbed the side of the building, walked over the roof, and then jumped down onto the goal. To find emergent behavior we can record a *typical* player trace which shows the expected path for a player, which can be used to compare with the player traces being evaluated. We could compare the player traces directly, however we are more interested in the player's general path through the environment. To calculate the general path of a player we need the world to be decomposed into a series of connected regions that can be navigated by the player (i.e., *negative space*). We can then compare the regions traversed by a player with the regions traversed by the typical player trace. If there is a high variance between the two traversal paths, we know that the player did not take the expected path. Finding these emergent behaviors can aid the designers in better understanding how players are interacting their environment and to discover short-cuts that the players might take that need to be addressed.

If a player plays the same level more than once or a similar level involving like challenges, we can track whether or not there is significant learning between their play sessions. If the player trace from their more recent play session is shorter than their previous one, it is considered *positive learning* (i.e., they actually learned from their prior experience). We can also use some of the other patterns to determine learning. For instance, if the player did fewer pirouettes than in their previous play session then that can be considered positive learning as well. Figure 3(e) shows an example of positive learning between two play sessions by the same player. In this example, the player was trapped by crates when they first started the level. In the first play session, they did not realize until much later that they can break the crates to escape. However, in their second play session they broke the crates immediately and escaped. The use of visual data mining can be used to track learning across levels and given challenges to assist in developing the desired experience.

5 IMPLEMENTATION

The game we used to apply this research is the Urban Combat Testbed project [Youngblood and Holder 2004; Cook et al. 2007; Youngblood et al. 2006]. The Urban Combat Testbed (UCT) is a first-person shooter based on the Quake 3 engine. A screenshot taken from UCT is shown in Figure 4. The main goal of a player in the UCT scenarios is to find and defuse an Improvised Explosive Device (IED) before the timer expires. Since we have several

thousand player traces in the UCT environment from previous experiments, we were able to tap into that data to run our visual data mining and information value experiments.



Figure 4: A screenshot of the Urban Combat Testbed Environment.

The interactive player visualization tool, known as PlayerViz, was implemented in C++ and OpenGL. A screenshot of PlayerViz is shown in Figure 5. The tool is able to load player traces and geometry in a format that is easily produced by most first and third person games. The user has several options when using PlayerViz, such as lighting, drawing the X, Y and Z axes, and drawing all negative space or just the bottom-most polygon (culling). The user can also swap Y and Z axes or Flip the X axis to conform with the standards used by their game. There is also a Reset Camera option to reinitialize the camera to its initial starting position. The user can load multiple player traces and selected traces are highlighted in the 3D view.

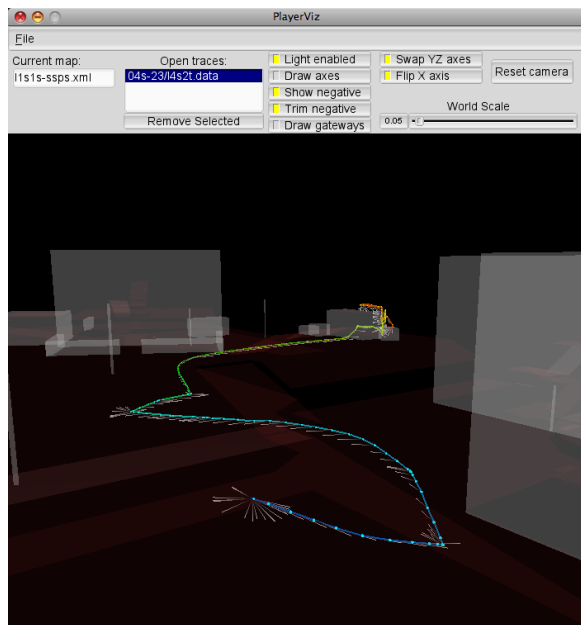


Figure 5: PlayerViz is an interactive tool for visual analysis of player log data.

Due to the open source and cross-platform libraries used, the tool will run on all major platforms including Linux, Mac OS X, and

Windows. The tool is now available as part of the Common Games Understanding and Learning (CGUL) toolkit at playground.uncc.edu/GameIntelligenceGroup/CGUL.

Even though we used UCT for our experimentation, our techniques can be applied to any data set captured from a first or third-person interactive environment as long it provides the necessary player information. The player traces contain the player's data over time such as x, y, z, yaw, pitch, roll, shots fired and so forth. The geometry is in an XML file format that contains the relevant surfaces and detailed geometry of the target environment. This file can be produced from any polyhedron-based 3D environment using a tool we call SARGE (SSPS Automated Region & Gateway Extractor) [Youngblood et al. 2006]. These files are in a well defined format that can be output by nearly any game, usually by the level designer. The Urban Combat Testbed is available for download at www.urban-combat.net. SARGE is also available with the CGUL toolkit at playground.uncc.edu/GameIntelligenceGroup/CGUL.

6 EXPERIMENTATION

We have a dataset of 3079 player traces collected from previous studies done in the UCT environment. We ran an experiment using this existing subject data to apply the visual data mining process. We wrote a program that would process each player trace we had and output four images of what the player trace looks like from different angles using PlayerViz. We then wrote a program that would create a series of web pages showing all of the player traces with a single trace per row and the different images on the columns. An example is shown in Figure 6. We then followed the visual data mining workflow outlined in Figure 7.

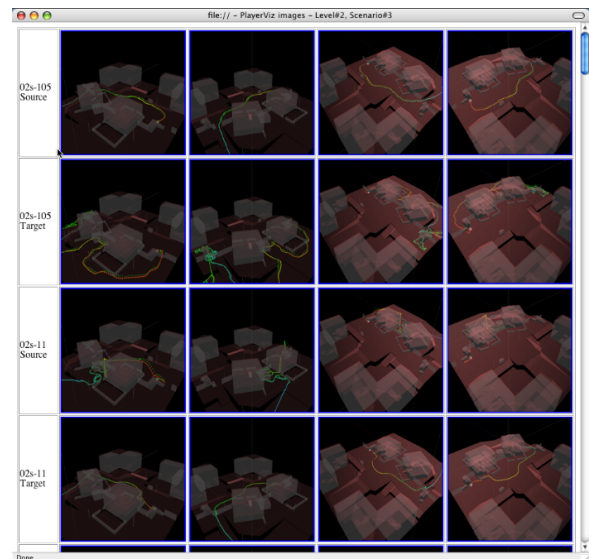


Figure 6: Webpage created to visually analyze a large collection of player traces.

6.1 Procedure

Once we had the collection of web pages, we searched for patterns by looking at the images and applying our domain knowledge. Once we found an interesting pattern, we recorded the ID of the player trace and continued the search. Once we found several patterns that we wanted to investigate, we wrote a program to automate the discovery of these patterns and search for them in the entire data

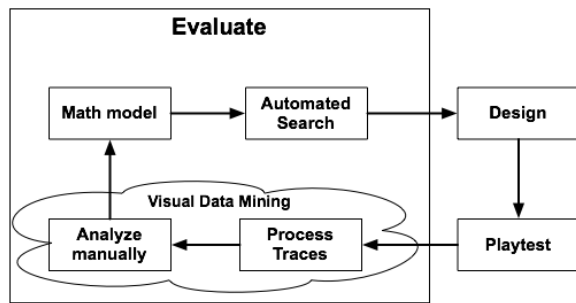


Figure 7: Process for visual data exploration and analysis of play test data.

set. The automated search tool, called *PlayerTraceProcessor*, processed every single player trace using the algorithms described in Section 4. Once it was finished processing the data (which was very quick), it produced a text file with detailed information about the patterns found including which player trace it was found in, how often it occurred, and where it occurred. We also generated a heat map to identify areas where some of the patterns were found most often.

6.2 Results

The patterns we searched for were: pirouettes (player rotates while standing still), flusters (loss of player view control), and jumpers (players that constantly jump). There were a total of 3079 player traces processed in the automated search. There were 291 pirouettes found in the data, 1023 flusters found, and 220 jumpers found. The processing was done in 108 seconds. Interestingly, 7.53% of player traces contained at least one pirouette with an average of 1.25 pirouettes per trace, and 7.15% of player traces were categorized as jumpers. 14.25% of player traces contained at least one fluster with an average of 2.33 flusters per trace. The maximum number of pirouettes for any one player trace was 6, while the maximum number of flusters for a trace was 46. The large amount of flusters indicates a problem with either the mouse sensitivity or the game’s rotation math.

We also recorded the locations where pirouettes and flusters occurred. Using this information we generated heat maps as seen in Figure 8. In figure 8(a) the spheres indicate where a pirouette occurred while the color indicates the density of pirouettes in that area (blue is low, red is high). The areas circled in white indicate where most of the pirouettes occurred. This shows us that most players perform a pirouette when they first start the game (top left) and while standing near the center of the map (bottom-center). We can also see the locations where most flusters occur as shown in Figure 8(b). The fluster heat map shows that most players lost control near the entrance to the top-left building and in the corner next to the bottom-left building. This was contrary to our expectations as we anticipated more flusters to occur inside the buildings as the players navigated in a more confined space.

7 CONCLUSION

To summarize our work, we will present an example usage of our process for a hypothetical single player first-person shooter game, that we shall refer to as *Shooter*. Suppose that *Shooter* is in development and a limited early version of the game is released to the public to play (known as a *public beta*). The game server could log all activity of players in the game environment. The data could

then be processed and an analyst could apply our visual data mining techniques. If it is found that many players perform a pirouette at a certain location in the level, the geometry of that location could be updated to have more distinctive features. A large number of flusters occurring in the game could indicate problems with the implementation of rotation math or mouse sensitivity. We could also compare multiple traces from the same player and examine whether they are solving challenges too quickly. If this is the case, we should present them with more difficult challenges. Emergent behavior could also be found through visual analysis. For instance, if many player circumvent a certain challenge in the level through some short-cut method, we could address this by eliminating the short-cut.

Logging of the player’s actions in real-time as they played *Shooter* could also help us dynamically adapt to their behaviors. For example, if a player performs a pirouette then we could instantly activate the navigation arrow that points in the direction they need to go. A fluster could also be corrected by rotating the player’s view to a stable state. If we detect that a player is jumping, we can highlight information artifacts more prominently to compensate for their increased speed and lack of environmental observation. The AI (artificially intelligent) agents could also adapt to the strategies of a player in real-time through the use of the logged data.

Through our experimentation we have shown that the visual representation of recorded game player data does provide knowledge about the player’s behavior in the game environment, which could then be used to guide design improvements. By visualizing our dataset of 3079 player traces we discovered several behavioral patterns and then automated the search to find more instances of these patterns in our dataset. We found that 7.53% of player traces contained pirouettes indicating areas where navigational aids are needed. Flusters were found in 14.25% of the player traces indicating control issues with the game engine. 7.15% of players can be clearly classified as jumpers because they constantly jump while exploring the environment indicating a need to provide additional attentional data to these players. Learning gains were also observed in some players when they play a scenario multiple times. We also noticed emergent player behavior that would be difficult to find without visual analysis.

The use of the *PlayerViz* tool provides the means for processing the player data in order to extract knowledge for experience improvement. With this knowledge, the designers can create interactive worlds that are adapted to the behavioral patterns of several different players instead of just the *ideal* player.

8 Future work

We plan to search for more patterns of behavior that we have noticed in our dataset. One such pattern is a *scan* which occurs when a player searches left and right and looks like a half-pirouette. This behavior is usually performed by experienced players who are very cautious about their surroundings. We also plan to incorporate our visual data mining pipeline into a multiplayer game to study how players interact with each other, competitively and cooperatively.

References

- BOFF, K. R., AND LINCOLN, J. E. 1988. *Engineering Data Compendium: Human Perception and Performance*. Wright-PattersonAFB.
- BRASHILL, J., AND BARNETT, J., 2007. Portal: In-game commentary. Valve Software, October.

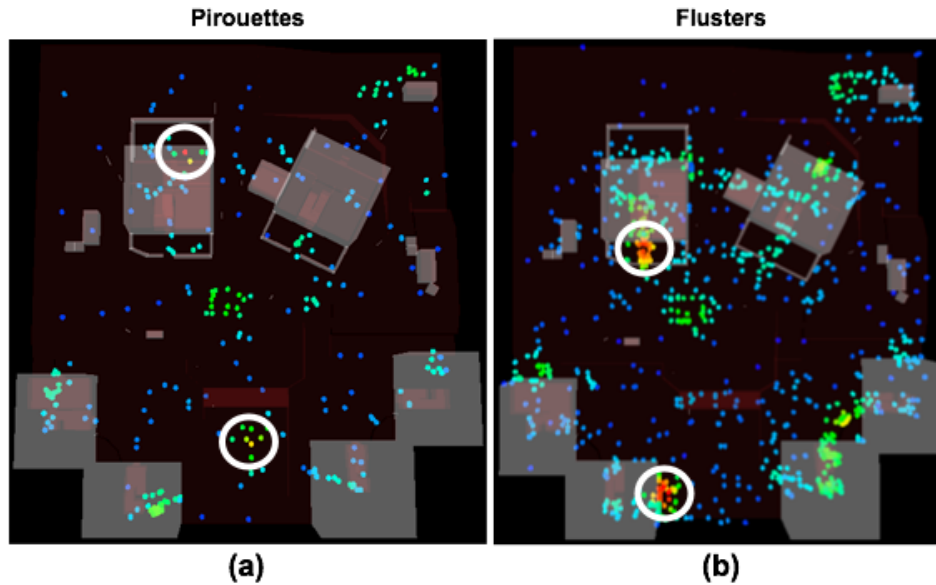


Figure 8: Heatmaps for (a) pirouettes and (b) flusters. The spheres indicate where a pattern occurred and the color indicates density of patterns in that area where blue is low and red is high. Areas circled in white are points of interest.

CHITTARO, L., RANON, R., AND IERONUTTI, L. 2006. VU-Flow: A Visualization Tool for Analyzing Navigation in Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics* 12, 6 (Nov/Dec), 1475–1485.

COOK, D. J., HOLDER, L. B., AND YOUNGBLOOD, G. M. 2007. Analysis of Human Transfer Learning Using a Real-Time Game Testbed. *IEEE Transactions on Knowledge and Data Engineering*. Under review.

DE OLIVEIRA, M. C. F., AND LEVKOWITZ, H. 2003. From Visual Data Exploration to Visual Data Mining: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 9, 3 (July/Sept), 378–394.

HERTEL, S., AND MEHLHORN, K. 1983. Fast Triangulation of the Plane with Respect to Simple Polygons. In *International Conference on Foundations of Computation Theory*.

HONG, Q., 2005. Question of the Week Responses: In-Game Advertising?, November. http://www.gamasutra.com/features/20051130/hong_01.shtml.

JIYANG CHEN, TONG ZHENG, W. T. O. R. Z., AND GOEBEL, R. 2007. Visual Data Mining of Web Navigational Data. *11th International Conference Information Visualization* 4, 6 (July), 649–656.

JOSLIN, S., BROWN, R., AND DRENNAN, P. 2007. The gameplay visualization manifesto: a framework for logging and visualization of online gameplay data. *Comput. Entertain.* 5, 3, 6.

KEIM, D. 2002. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (March).

MARSELAS, H., 2000. Profiling, Data Analysis, Scalability, and Magic Numbers, Part I: Meeting the Minimum Requirements for Age of Empires II: The Age of Kings, August. http://www.gamasutra.com/features/20000809/marselas_01.htm.

MEYER, R. D., AND COOK, D. 2000. Visualization of data. *Current Opinion in Biotechnology* 11, 1 (February), 89–96.

NIEDENTHAL, S. 2007. Real-Time Sweetspot: The Multiple Meanings of Game Company Playtests. *Proceedings of DiGRA 2007 Conference*, 697–702.

SOFTWARE, V., 2007. Half-Life 2: Episode Two Stats. http://www.steamgames.com/status/ep2/ep2_stats.php.

THOMPSON, C. 2007. The science of play. *Wired* 15, 9, 140–147,184.

TRACY FULLERTON, C. S., AND HOFFMAN, S. 2004. *Game Design Workshop*. CMP Books.

YOUNGBLOOD, G. M., AND HOLDER, L. B. 2004. Agent-based Players for a First-person Entertainment-based Real-time Artificial Environment. In *the 17th International Conference of the Florida Artificial Intelligence Research Society (FLAIRS) held in Miami Beach, Florida*.

YOUNGBLOOD, G. M., NOLEN, B., ROSS, M., AND HOLDER, L. 2006. Building Test Beds for AI with the Q3 Mod Base. In *Artificial Intelligence in Interactive Digital Entertainment (AIIDE)*.