CSE 561: Modeling & Simulation Theory and Application

Deepak Reddy Nayani
School of Computing and Augmented Intelligence
Arizona State University
Tempe, Arizona, USA
dnayani@asu.edu

***Abstract*—In any online game, it is imperative that the players have an enjoyable time playing the game. This is assured by matching similarly skilled players as teammates and opponents and ensuring fairness in every match. Usually, an online multiplayer game requires the player to join a waiting queue. The matchmaking system then arranges for the player to enter an impromptu game with similarly skilled players.**

The central premise of the simulation is to develop a queuing system that matches 10 players, 5 on each side, of varying (but reasonably similar) skill levels and aims to maximize fairness and minimize wait times. This is a discrete event simulation. The experimentation will be done on modular and coupled subcomponents, which are part of a well-defined hierarchy. Each subcomponent has its own specification and purpose, i.e., controlling the queuing time distribution, logging activity and times, maintaining and processing the queues etc. The queuing algorithm is an efficient linear time greedy algorithm that matches the 10 players. The scope of the simulator is not only to create matches but to retrieve and store important data, ranging from average wait times to players' behavioural information.

I. Introduction

This is a portfolio report for the Modeling and Simulation Theory and Application (CSE 561) course in Fall 2022. We developed a project, namely, *Simulating Matchmaking in Online Multiplayer Games*. As part of the project, we developed a high-level model of a matchmaking scenario of online gaming using Java, Maven and the DEVSuite Simulation Framework, simulated the model and carried out experiments with test cases to check its efficacy and how it operates concerning logical time.

A. Background

Online gaming is one of the most prevalent forms of recreation in today's world for over 80% per cent of today's Gen-Z and millennial generation, of which 65% per cent are casual players [1]. The most important motivator for any person to continue playing a game is to find it 'fun.' This is ensured by ensuring that every single game they compete in is fair and not one-sided, which is achieved by stacking players with similar skill metrics to allow new joining players to play with similarly skilled players and provide an even chance of progression as they continue playing.

Moving on to the project's scope, the idea is to simulate a queuing system that accounts for the predefined skill and the player's behaviour. Each player possesses a certain skill level and tolerance for waiting in the queue. A hierarchical model with subcomponents, including a control unit, queuing system, etc., is specified. We explore each subcomponent in the later sections.

For this simulation, we do not delve into the entire depth and mechanism of an online gaming queuing system. The idea is to keep in mind a high-level concept of matchmaking that can be simulated to depict the challenges of not making the player wait too long for a match but simultaneously making the player feel that the match was fair. The simulation abstracts away the low-level details like server-client latency, age and gender of player, changes in player traffic depending on the time of the day, etc., in the queuing process, as these details are not pertinent to the aspects of player satisfaction; we intend to analyze.

B. Problem Definition

The purpose of any queuing model is to observe and study the players' behaviour and wait times and how the system's various statistical aspects vary as we tweak various system parameters. In a similar fashion, even in the concept of online games, players must wait in queues to get into their games. The question is now, how can we observe these queuing times and player behaviour? By simulating the system model of the queue whilst considering the player's skill rank and patience and finding an optimal average time for every match that is created, all while maintaining a consistent skill range for any given match, we can look at how each subcomponent in the system interacts with one another to deal with various real-life constraints such as queue sizes, extremely long wait times due to inadequate player pool or incompatible skill levels etc. Most contemporary online multiplayer video games employ a FIFO (First-in First-out) queue to regulate the flow of incoming players looking to play. A study [2] claims that the FIFO model significantly outperforms any other queuing order.

Given the actual complexities of incoming player modeling and queuing modeling, a set of assumptions are put in place to simplify the problem but keep it scalable to enhance the algorithm and other variables that influence the queue wait times of the players. These assumptions abstract away the low-level details, which we can ignore because they do not affect the analysis and the study.

- When a player requests the server to arrange a match, the system puts them into a queue corresponding to their assigned skill level. Thus, each queue has multiple players of the skill levels that correspond to the skill tier of the queue, waiting to start a match.
- Once enough players are aggregated, the algorithm selects 10 similarly skilled players, and they will be put into a match and dequeued from the matchmaking system.
- We assume the simulation to have a finite number of players at the beginning and do not consider the recurrence of the players who have already played. Generally, a simulation will run until the initial player pool is empty.
- Additional player behaviour, such as players opting out of the queue for any reason, is depicted using a single threshold (the patience level, in time units), after which the player will remove themselves from the queue.
- A FIFO queuing order is followed, i.e., players are matched and outputted as and when a new compatible player joins the queue.
- An overall time threshold is assumed for each skill queue, after which it will allocate for cross-tier matchmaking operations (whilst ensuring the match is still as fair as possible).
- Any player that arrives when a queue is full is discarded. The player then stays dormant for the rest of the simulation.

II. SYSTEM DESCRIPTION

The system has two primary entities: the central game server and the players.

A. Server

The server stores all the matchmaking logic and has the unique responsibility of grouping individual players into teams to arrange for a game match. A game match generally consists of multiple rounds, which is not a part of the model or the simulations. After the match is over, the player has effectively exited the system. Now, the player can again choose to join the queue to play another match. However, in the simulations, this is forbidden; hence, one player only gets to play one match at most. In other words, the players quit and do not re-engage with the game after playing one match.

The server considered for this problem is a single consolidated server, which takes in or accepts the incoming player into it and sends the player into one of the 3 skill-tier queues. The scope for the skill level in this system/model is limited to 3 skill tiers, with each tier consisting of 3 (sub-)skills. There are three skill tiers: 1, 2, and 3. Each skill tier has three skills, as follows:

- Tier 1 contains skills 1, 2, 3
- Tier 2 contains skills 4, 5, 6
- Tier 3 contains skills 7, 8, 9

To expand the scope of skills in future work, one can change the range of skill values from integers to real numbers between 1 and 9. The idea is to keep the model scalable and give the server operators or game developers an option to have different values for said skill levels based on more sophistical research into player gameplay and statistics. This would allow the players to be assigned a more precise and high-granularity skill value which could increase the quality of matchmaking fairness.

B. Player

The following assumptions have been made regarding the behaviour of each player in the model:

- A player has a fixed skill level, and it does not change at any point.
- A player stays dormant:
  - before arriving at the queue.
  - before requesting to play a game match.
  - while waiting in the queue for a match to be arranged.
  - after the match is finished.
  - after getting their request to play a match rejected because the server is busy/full.
  - after giving up and disconnecting due to too long of a wait time.
- A player participates in only one game at a time.
- The player does not get to choose which queue they join.
- While waiting in the queue, if a player reaches their patience threshold, they disconnect from the server and stay dormant.
- A player does not quit or disconnect from the server for any reason other than reaching their patience threshold, such as a power outage, network connectivity issues, some other work/appointment, etc.
- The satisfaction perceived by a player solely depends on two factors only:
  - total time spent in the queue waiting for a match to begin (not counting the time spent before joining the queue).
  - the fairness of the match (judged by the distribution of skill levels of players in the two teams).
- The satisfaction perceived by a player does NOT depend on other factors such as the outcome (win/loss) of the match, their performance in the match, interaction quality with other players, etc.
- Only a single player ever arrives at the queue, i.e. enqueuing 2 or more players at a time is not possible in the model.
- The average traffic of players does not change with the time of the day or the day of the year.

A player has been modelled as an entity which has the following attributes:

- id (the fixed unique ID assigned to the player)
- skill (the fixed skill level assigned to the player)
- patience (the threshold which results in the player giving up and quitting/disconnecting)
- joinTime (the precise time at which the player joined the queue to wait in it)

III. MODELING

In this report, the following notations are used:

- $\mathbb{Z}^+$ is the set of positive integers (zero is excluded).
- $\mathbb{Z}_0^+$ is the set of non-negative integers (zero is included).
- $\mathbb{R}^+$ is the set of positive real numbers (zero is excluded).
- $\mathbb{R}_0^+$ is the set of non-negative real numbers (zero is included).
- $[x_1, x_2, x_3, ..., x_n]$ is an ordered sequence of n items $x_1, x_2, x_3, ..., x_n$

A. Model Description

Before moving into the model specification, it is imperative to understand the data types of the various player objects, variables, states, etc., used in the model. The model uses several data types, including primitive data types such as strings and integers and complex data types such as player objects and queues. The patience threshold of a player is a string, and a player's rank is an integer. A player object consists of a patience threshold (a string) and a rank (an integer). A queue is a list of player objects and is thus a complex data type.

The state variable S = phase x R+ is a complex data type consisting of a phase (a primitive string type) and an actual number (a primitive float/double data type).

The system follows a single-server single-stage queue. The queue considered for the matchmaking process is a Poisson queue, and the arrival of players into the system is independent of one another. It will, thus, follow **Poisson Process**.

$$Arrivals \longrightarrow Queue \longrightarrow Service \longrightarrow Done$$

The number of players arriving in a unit length of time is given by:

$$Prob(Arrivals = x) = \frac{e^{-\lambda t}(\lambda t)^x}{x!}$$

The inter-arrival time gap between any two players (with no other player arriving in between them) is given by:

$$Prob(Delay = t) = \lambda e^{-\lambda t}$$

1) Player Entity Generator: The Player Entity Generator (PEG) is responsible for creating player entities with their assigned skill and patience threshold, and PEG outputs the generated players to the Matchmaking Handler (MMH) (refer to Figure 1). PEG allows us to generate fake players with chosen distributions of arrival times, skill levels, and patience thresholds. Similar to how players would arrive and get managed by the MMH in a real-world scenario, the fake players generated by PEG get injected into the system and managed by MMH. As mentioned before, the arrival distribution of the players is assumed to be a Poisson process, so the inter-arrival time gaps are exponentially distributed, but the parameter $lambda$ which affects the average inter-arrival time gap can still be configured. The skill levels and patience thresholds can be tested with a uniform or a normal distribution.

2) Matchmaking Handler: Referring to Figure 1, the Matchmaking Handler (MMH) is an essential subcomponent of the system that regulates the influx of players into the server. Based on the skill level, it enqueues the player into the respective tier queue (Q_1, Q_2 or Q_3). However, some constraints must be met for the player to be accepted into the queue, e.g., the server queue must not be complete. The MMH can observe the states (e.g. whether complete or not) of the queues Q_1, Q_2 and Q_3 via the feedback into its ports "$in1$", "$in2$", and "$in3$", to facilitate said regulation of players. The state of MMH is always "active".

3) The Three Queues: Referring to Figure 1, the queue subcomponents receive the player entity and let the player wait until a match can be arranged. As mentioned earlier, each queue is dedicated to its corresponding skill tier. Specifically, Q_1 enqueues players with skill tier 1 (skills 1-3), Q_2 with skill tier 2 (skills 4-6) and Q_3 with skill tier 3 (skills 7-9). Upon satisfying the matching criterion, the players are dequeued and transferred into the Team Assembler (TA) subcomponent. When the player composition in the queue is such that a match cannot be arranged among the players in the same tier, and a long time has passed waiting in the queue (judged by a configurable threshold parameter), then the algorithm expands its scope and increases the player pool by merging two different queues together. In this case, the algorithm has a better chance of finding a match such that the two teams are still relatively evenly matched in terms of the skills of the individual players. It is essentially trading a bit of fairness for better wait times.

4) Team Assembler: Referring to Figure 1, the arranged match, consisting of 10 player outputs from the queue, is sent to the Team Assembler. The Team Assembler is responsible for initiating the match. We assume that the time taken to create a match is instantaneous, i.e., the response time is 0. Additionally, various information about the arranged match (the roster of the two teams, the player IDs, the player's skills, the wait time of each player, etc.) is fed into the Activity Logger for later analysis.

5) Activity Logger: Referring to Figure 1, the Activity Logger is essentially the information hub of the system, storing data ranging from player details, team details, average wait time for players, disconnected players, log of all timed events, overall judged skill of team (can be used as a fairness check), etc.

B. Model Specification

1) Fundamental Entities Definitions:

$$Player = \mathbb{Z}_0^+ \times \{1,2,3\} \times \mathbb{R}^+ \times \mathbb{R}_0^+$$

where id $\in \mathbb{Z}_0^+$, skill $\in$ {1,2,3}, patience $\in \mathbb{R}^+$, joinTime $\in \mathbb{R}_0^+$

$$Match = \{p_1, p_2, p_3, ..., p_{10}\}$$

where p_1, p_2, p_3, p_4, p_5 are 5 players in the first team and $p_6, p_7, p_8, p_9, p_{10}$ are 5 players in the second team.

$$Skill = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

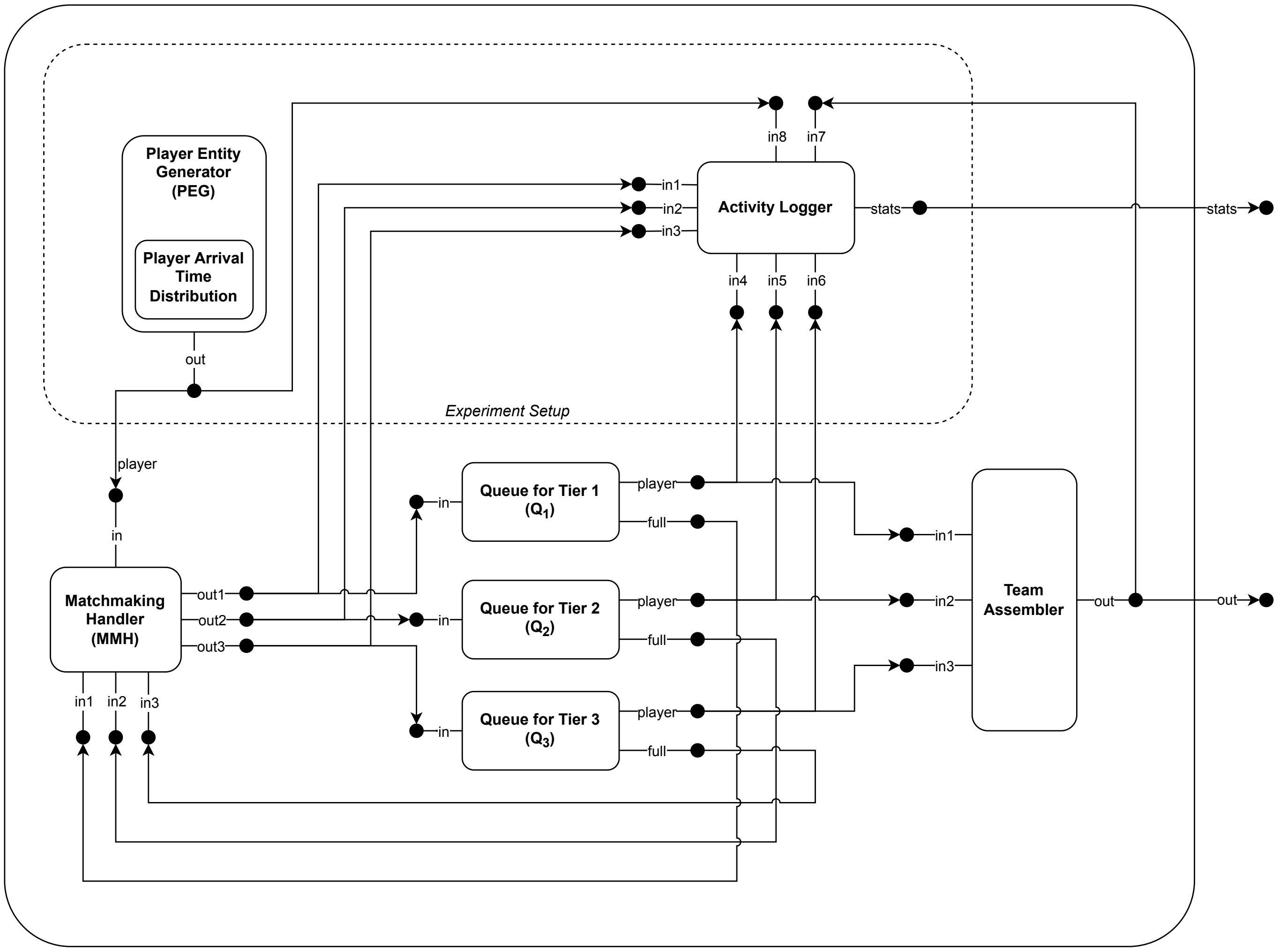

Fig. 1. Schematic Model Subcomponents Interconnection Diagram

$$Tier = \{T_1, T_2, T_3\}$$

where each tier contains the skills as shown: $T_1 = \{1, 2, 3\}$, $T_2 = \{4, 5, 6\}$, and $T_3 = \{7, 8, 9\}$
$Q_i = [p_1, p_2, p_3, ..., p_n]$ is a queue of n players, each player having skill i where $i \in Skill, \quad p_1, p_2, p_3, ..., p_n \in Player.$

$$Statistics = \{\mathbb{R}_0^+, \mathbb{Z}_0^+, \mathbb{Z}_0^+, \mathbb{Z}_0^+\}$$

is a tuple of the following numerical measurements:

- $AvgWaitTime$, the average amount of time spent by players in the waiting queue
- $matchedTeamsCount$, the number of matches successfully created/arranged
- $quitPlayers$, the number of players who quit after joining the waiting queue
- $rejectedPlayers$, the number of players who got rejected and didn't join the waiting queue

2) Top-level DEVS Coupled Model Definitions:

$$N = \langle X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC\rangle$$

$InPorts = \{\text{“}in\text{”}\}$
$X_{in} = \{x \mid x \ \in Player\}$
$X = \{(p, v) \mid p \in InPorts, v \in X_{in}\}$
$OutPorts = \{\text{“}out\text{”}, \text{“}stats\text{”}\}$
Y_{out} = { y | y $\in$ Match }
Y_{stats} = { y | y $\in$ Statistics }
Y = { (p, v) | p $\in$ OutPorts, v $\in$ Y_{out} }
D is the set of components. $\forall$ d $\in$ D, M_d is a subcomponent of the top-level model N.
$M_d = \langle X_{M_d}, Y_{M_d}, S_{M_d}, \delta^{ext}_{M_d}, \delta^{int}_{M_d}, \delta^{con}_{M_d}, \lambda_{M_d}, ta_{M_d}\rangle$ is a DEVS model, having:

$$\begin{aligned} InPorts_{M_d} &= \{\text{“}in1\text{”}, \text{“}in2\text{”}, \text{“}in3\text{”}, ...\} \\ X_{M_d} &= \{(p, v) \mid p \in InPorts_{M_d}, v \in X_p\} \\ OutPorts_{M_d} &= \{\text{“}out1\text{”}, \text{“}out2\text{”}, \text{“}out3\text{”}, ...\} \\ Y_{M_d} &= \{(p, v) \mid p \in OutPorts_{M_d}, v \in Y_p\} \end{aligned}$$

EIC $\subseteq$ { ((N, IP_N), (M_d, IP_{M_d})) | IP_N $\in$ InPorts, d $\in$ D, $IP_{M_d} \in InPorts_{M_d}$ }
EOC $\subseteq$ { ((M_d, OP_{M_d}), (N, OP_N)) | $OP_N \in OutPorts$, d $\in$ D, $OP_{M_d} \in OutPorts_{M_d}$ }

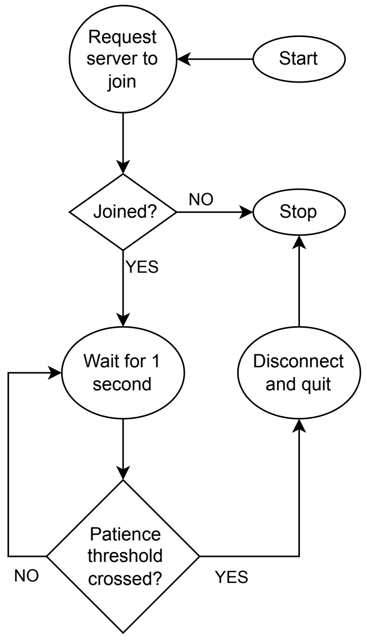

Fig. 2. Player Flow Chart

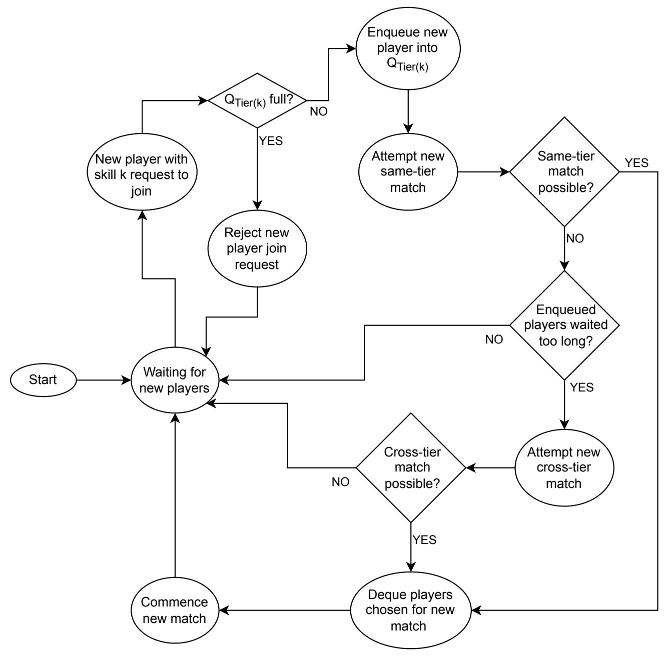
Enqueue new player into $Q_{Tier(k)}$
$Q_{Tier(k)}$ full?
NO
YES
New player with skill k request to join
Attempt new same-tier match
Same-tier match possible?
YES
Reject new player join request
NO
Enqueued players waited too long?
NO
Start
Waiting for new players
YES
Cross-tier match possible?
Attempt new cross-tier match
NO
YES
Commence new match
Deque players chosen for new match

Fig. 3. Server Flow Chart

IC $\subseteq$ { $((M_a, OP_{M_a}), (M_b, IP_{M_b})) \mid OP_{M_a} \in OutPorts_{M_a}$, a, b $\in$ D, $IP_{M_b} \in InPorts_{M_b}$ }

3) Subcomponents (DEVS Coupled Models) and Interconnections:

The following are the components involved in this model:

- N is the top-level DEVS model.
- M_{PEG} is the DEVS model for Player Entity Generator.
- M_{MMH} is the DEVS model for Matchmaking Handler.
- M_{Q_1} is the DEVS model for the Tier-1 queue Q_1.
- M_{Q_2} is the DEVS model for the Tier-2 queue Q_2.
- M_{Q_3} is the DEVS model for the Tier-3 queue Q_3.
- M_{LOG} is the DEVS model for the Activity Logger.
- M_{ASS} is the DEVS model for the Team Assembler.

The following interconnections are in place between the models:

- $(N, \text{“}in\text{”}) \longrightarrow (M_{MMH}, \text{“}in\text{”})$
- $(M_{ASS}, \text{“}out\text{”}) \longrightarrow (N, \text{“}out\text{”})$
- $(M_{LOG}, \text{“}stats\text{”}) \longrightarrow (N, \text{“}stats\text{”})$
- $(M_{PEG}, \text{“}out\text{”}) \longrightarrow (M_{MMH}, \text{“}in\text{”})$
- $(M_{Q_1}, \text{“}full\text{”}) \longrightarrow (M_{HMM}, \text{“}in1\text{”})$
- $(M_{Q_2}, \text{“}full\text{”}) \longrightarrow (M_{HMM}, \text{“}in2\text{”})$
- $(M_{Q_3}, \text{“}full\text{”}) \longrightarrow (M_{HMM}, \text{“}in3\text{”})$
- $(M_{HMM}, \text{“}out1\text{”}) \longrightarrow (M_{Q_1}, \text{“}in\text{”})$
- $(M_{HMM}, \text{“}out2\text{”}) \longrightarrow (M_{Q_2}, \text{“}in\text{”})$
- $(M_{HMM}, \text{“}out3\text{”}) \longrightarrow (M_{Q_3}, \text{“}in\text{”})$
- $(M_{Q_1}, \text{“}player\text{”}) \longrightarrow (M_{ASS}, \text{“}in1\text{”})$
- $(M_{Q_2}, \text{“}player\text{”}) \longrightarrow (M_{ASS}, \text{“}in2\text{”})$
- $(M_{Q_3}, \text{“}player\text{”}) \longrightarrow (M_{ASS}, \text{“}in3\text{”})$
- $(M_{MMH}, \text{“}out1\text{”}) \longrightarrow (M_{LOG}, \text{“}in1\text{”})$
- $(M_{MMH}, \text{“}out2\text{”}) \longrightarrow (M_{LOG}, \text{“}in2\text{”})$
- $(M_{MMH}, \text{“}out3\text{”}) \longrightarrow (M_{LOG}, \text{“}in3\text{”})$
- $(M_{Q_1}, \text{“}player\text{”}) \longrightarrow (M_{LOG}, \text{“}in4\text{”})$
- $(M_{Q_2}, \text{“}player\text{”}) \longrightarrow (M_{LOG}, \text{“}in5\text{”})$
- $(M_{Q_3}, \text{“}player\text{”}) \longrightarrow (M_{LOG}, \text{“}in6\text{”})$
- $(M_{ASS}, \text{“}out\text{”}) \longrightarrow (M_{LOG}, \text{“}in7\text{”})$
- $(M_{PEG}, \text{“}out\text{”}) \longrightarrow (M_{LOG}, \text{“}in8\text{”})$

So, finally, for the top-level DEVS model:
$EIC = \{((N, \text{“}in\text{”}), (M_{MMH}, \text{“}in\text{”}))\}$
$EOC = \{((M_{ASS}, \text{“}out\text{”}), (N, \text{“}out\text{”})),$
$((M_{LOG}, \text{“}stats\text{”}), (N, \text{“}stats\text{”}))\}$
$IC = \{$
$((M_{PEG}, \text{“}out\text{”}), (M_{MMH}, \text{“}in\text{”})),$
$((M_{Q_1}, \text{“}full\text{”}), (M_{HMM}, \text{“}in1\text{”})),$
$((M_{Q_2}, \text{“}full\text{”}), (M_{HMM}, \text{“}in2\text{”})),$
$((M_{Q_3}, \text{“}full\text{”}), (M_{HMM}, \text{“}in3\text{”})),$
$((M_{HMM}, \text{“}out1\text{”}), (M_{Q_1}, \text{“}in\text{”})),$
$((M_{HMM}, \text{“}out2\text{”}), (M_{Q_2}, \text{“}in\text{”})),$
$((M_{HMM}, \text{“}out3\text{”}), (M_{Q_3}, \text{“}in\text{”})),$
$((M_{Q_1}, \text{“}player\text{”}), (M_{ASS}, \text{“}in1\text{”})),$
$((M_{Q_2}, \text{“}player\text{”}), (M_{ASS}, \text{“}in2\text{”})),$
$((M_{Q_3}, \text{“}player\text{”}), (M_{ASS}, \text{“}in3\text{”})),$
$((M_{MMH}, \text{“}out1\text{”}), (M_{LOG}, \text{“}in1\text{”})),$
$((M_{MMH}, \text{“}out2\text{”}), (M_{LOG}, \text{“}in2\text{”})),$
$((M_{MMH}, \text{“}out3\text{”}), (M_{LOG}, \text{“}in3\text{”})),$
$((M_{Q_1}, \text{“}player\text{”}), (M_{LOG}, \text{“}in4\text{”})),$
$((M_{Q_2}, \text{“}player\text{”}), (M_{LOG}, \text{“}in5\text{”})),$
$((M_{Q_3}, \text{“}player\text{”}), (M_{LOG}, \text{“}in6\text{”})),$
$((M_{ASS}, \text{“}out\text{”}), (M_{LOG}, \text{“}in7\text{”})),$
$((M_{PEG}, \text{“}out\text{”}), (M_{LOG}, \text{“}in8\text{”})),$
$\}$

4) Queue Subcomponent Model Definitions:

$M_{Q_i} = \langle X,\ Y,\ S,\ \delta_{ext},\ \delta_{int},\ \delta_{con},\ \lambda,\ ta \rangle$
$InPorts = \{\text{“}in\text{”}\}$
$X_{in} = \{x \mid x \in Player\}$
$X = \{(p, v) \mid p \in InPorts,\ v \in X_p\}$
$OutPorts = \{\text{“}player\text{”}, \text{“}full\text{”}\}$
$Y_{player} = \{y \mid y \in Player\}$
$Y_{full} = \{0, 1\}$
$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$
$S = \{\text{“}active\text{”}, \text{“}full\text{”}, \text{“}respond\text{”}\} \times \mathbb{R}_0^+ \times Q_i$
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$
$\lambda(\text{“}respond\text{”}, \sigma, Q_i) = (\text{“}player\text{”}, y)$, where $y \in Player$
$\lambda(\text{“}active\text{”}, \sigma, Q_i) = (\text{“}full\text{”}, 0)$
$\lambda(\text{“}full\text{”}, \sigma, Q_i) = (\text{“}full\text{”}, 1)$
$ta(phase, \sigma, Q_i) = \sigma$

5) Matchmaking Handler Subcomponent Model Definitions:

$M_{MMH} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$
$InPorts = \{\text{“}in\text{”}, \text{“}in1\text{”}, \text{“}in2\text{”}, \text{“}in3\text{”}\}$
$X_{in} = X_{in1} = X_{in2} = X_{in3} = \{x \mid x \in Player\}$
$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$
$OutPorts = \{\text{“}out1\text{”}, \text{“}out2\text{”}, \text{“}out3\text{”}\}$
$Y_{out1} = Y_{out2} = Y_{out3} = \{y \mid y \in Player\}$
$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$
$S = \{\text{“}active\text{”}\} \times \mathbb{R}_0^+$
$\delta_{ext}(phase, \sigma, e, x) = (phase, \sigma)$
$\delta_{int}(phase, \sigma) = (phase, \sigma)$
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$

6) Player Entity Generator Subcomponent Model Definitions:

$M_{PEG} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$
$InPorts = X = \phi$
$OutPorts = \{\text{“}out\text{”}\}$
$Y_{out} = \{y \mid y \in Player\}$
$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$
$S = \{\text{“}passive\text{”}, \text{“}respond\text{”}\} \times \mathbb{R}_0^+$
$\delta_{ext}(, \sigma, e, x) = \phi$
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$
$\lambda(\text{“}respond\text{”}, \sigma) = (\text{“}out\text{”}, y)$, where $y \in Player$
$ta(phase, \sigma) = \sigma$

7) Team Assembler Handler Subcomponent Model Definitions:

$M_{ASS} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$
$InPorts = \{\text{“}in1\text{”}, \text{“}in2\text{”}, \text{“}in3\text{”}\}$
$X_{in1} = X_{in2} = X_{in3} = \{x \mid x \in Player\}$
$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$
$OutPorts = \{\text{“}out\text{”}\}$
$Y_{out} = \{(y_1, y_2, y_3, ..., y_{10}) \mid y_i \in Player\}$
$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$
$S = \{\text{“}active\text{”}\} \times \mathbb{R}_0^+$
$\delta_{ext}(phase, \sigma, e, x) = (phase, \sigma)$
$\delta_{int}(phase, \sigma) = (phase, \sigma)$
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$
$\lambda(\text{“}active\text{”}, \sigma) = (\text{“}out\text{”}, (y_1, y_2, y_3, ..., y_{10}))$ where $y_i \in Player$
$ta(phase, \sigma) = \sigma$

8) Activity Logger Sub component Model Definitions:

$M_{LOG} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$
$InPorts = \{\text{“}in1\text{”}, \text{“}in2\text{”}, \text{“}in3\text{”}, \text{“}in4\text{”}, \text{“}in5\text{”}, \text{“}in6\text{”}, \text{“}in7\text{”}, \text{“}in8\text{”}\}$
$X_{in1} = X_{in2} = X_{in3}, X_{in4} = X_{in5} = X_{in6} = X_{in8} = \{x \mid x \in Player\}$
$X_{in7} = \{(x_1, x_2, x_3, ..., x_{10}) \mid x_i \in Player\}$
$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$
$OutPorts = \{\text{“}stats\text{”}\}$
$Y_{stats} = \{y \mid y \in Statistics\}$
$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$
$S = \{\text{“}active\text{”}\} \times \mathbb{R}_0^+$
$\delta_{ext}(phase, \sigma, e, x) = (phase, \sigma)$
$\delta_{int}(phase, \sigma) = (phase, \sigma)$
$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$
$\lambda(\text{“}active\text{”}, \sigma) = \phi$
$ta(phase, \sigma) = \sigma$

C. Model implementation

The model has been implemented using the DEVS-Suite [3], [4] software in Java language. The model implementation “looks” different from the specified model because of pragmatic reasons, such as:

- $stats$ output in the Activity Logger has been replaced by 4 individual outputs ($AvgWaitTime$, $matchedTeamsCount$, $quitPlayers$, $rejectedPlayers$), which emit single numbers, as this is easier to implement in DEVS-Suite and looks more intuitive when visualized using SimView.
- The Team Assembler (ASS) component has been omitted, and the output port $full$ of the Queue components (Q1, Q2, Q3) have been omitted. Implementing the complex cross-tier matchmaking logic between the components MMH, Q1, Q2, Q3 and ASS in DEVS-Suite would be too complicated if done with pure message-passing. Hence, apropos of the project scope and deadlines, the complex cross-tier matchmaking logic directly accesses the required variables inside the involved components, obviating the need for the omitted parts. Figure 4 shows the complete hierarchical model as shown in SimView in DEVS-Suite.

1) Player entities: For generating the player entities, the following distributions were used:

- $skill$ — uniform distribution over $\{1, 2, 3, ..., 9\}$
- $patience$ — uniform distribution over $[a, b]$, where $a, b \in \mathbb{R}_0^+$ and $a < b$.

The inter-arrival time between two players is, of course, exponentially distributed. For message-passing between compo-

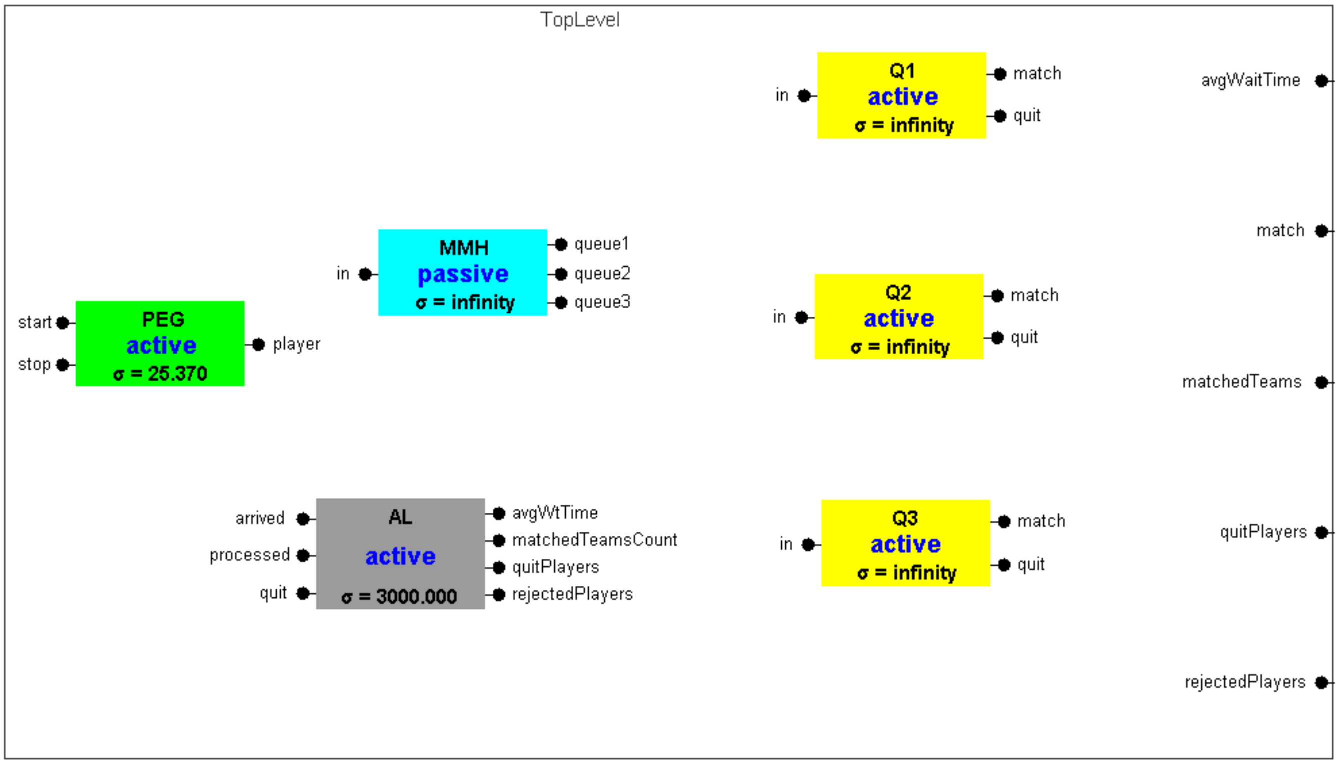

Fig. 4. DEVS-Suite SimView

nents, the serialization scheme was developed, which allowed transforming any player entity (including all its information fields) into a string and back from the string to the player entity object.

2) Matchmaking Logic: For the matchmaking logic, please refer to Fig. 2 and Fig. 3.

IV. EXPERIMENTS DESCRIPTIONS

In all of the following experiments, the following parameters are fixed:

- Size of all three queues is 15 players unless specified otherwise in the experiment.
- The patience threshold of a player is infinite by default unless specified otherwise in the experiment.
- The threshold for triggering a cross-tier match instead of the same-tier match is infinite unless specified otherwise in the experiment.

Experiment 1

Scenario: All queues are empty.
Input: Inject a player with skill 2
Expected Output: The player should get enqueued into Q_1. The phase of Q_1 should stay "active".

Experiment 2

Scenario: All queues have a capacity of 5 players. Q_1 and Q_2 are empty. Q_3 has 4 players.
Input: Inject a player with skill 7
Expected Output: The player should get enqueued into Q_3. The phase of Q_3 should change from "active" to "full".

Experiment 3

Scenario: All queues have a capacity of 5 players. Q_1 and Q_2 are empty. Q_3 has 4 players.
Input: Inject two players with skill 7
Expected Output: The player who was injected first should get enqueued into Q_3 and the phase of Q_3 should change from "active" to "full". The player injected second should get discarded, and the contents of Q_3 should not change in any way.

Experiment 4

Scenario: All queues are empty.
Input: Inject one player of each skill, i.e. 9 players, each having a unique skill ranging from 1 to 9.
Expected Output: 3 players should get enqueued into Q_1, three into Q_2 and three into Q_3. The phases of all three queues should stay "active".

Experiment 5

Scenario: All queues are empty.
Input: Inject 10 players with skill 1
Expected Output: When 10^{th} player is injected, a new match should be created with all 10 players. Afterwards, all queues should be empty again.

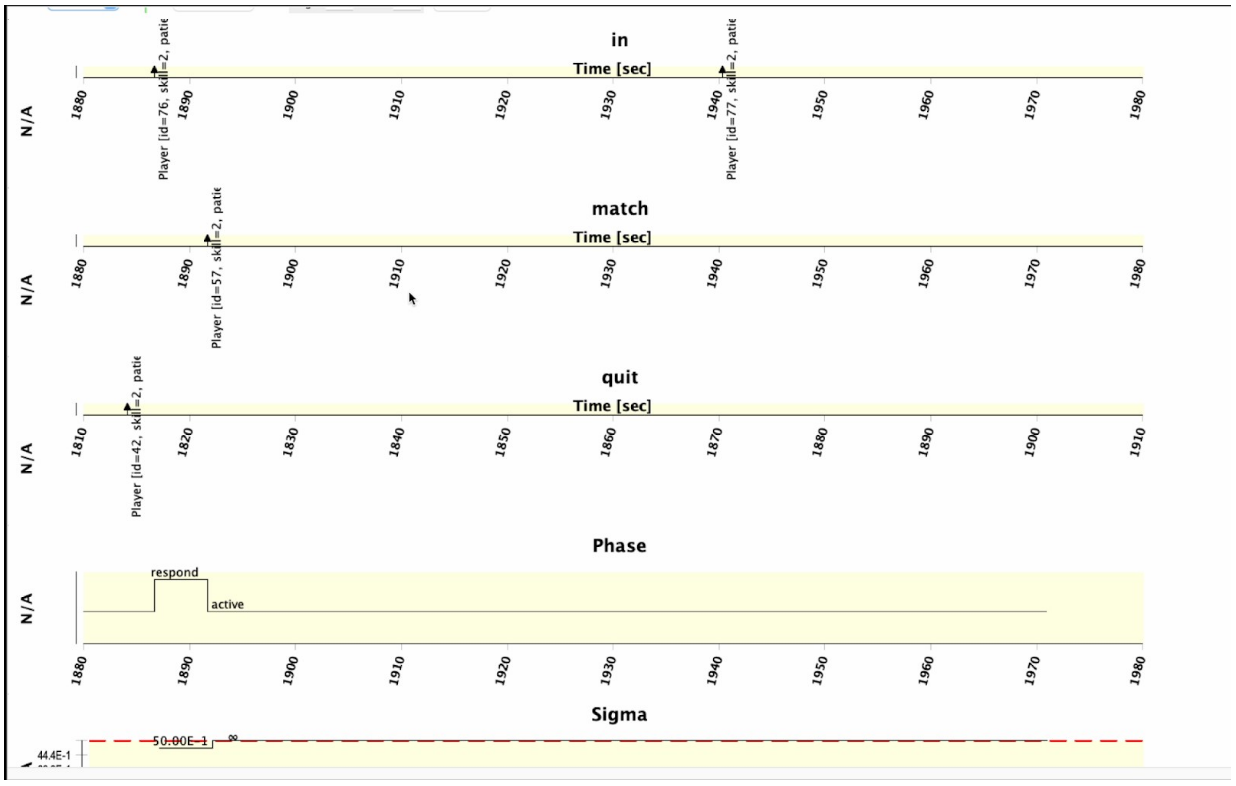

Fig. 5. Phase change when match is created

Experiment 6

Scenario: All queues are empty.

Input: Inject 50 players with skill 4

Expected Output: When 10^{th}, 20^{th}, 30^{th}, 40^{th}, and 50^{th} player is injected, each time a new match should be created with the last 10 enqueued players.

Experiment 7

Scenario: All queues are empty.

Input: Inject 9 players with a patience threshold of 1 minute and skill 3.

Expected Output: 1 minute after the 9^{th} player is injected, all server queues should be empty, and all 9 players should be disconnected.

Experiment 8

Scenario: The capacity of all three queues is 8 players. All queues are empty. The threshold for triggering cross-tier matchmaking is 10 minutes.

Input: Inject a player with skill 2 at 6:00 pm. Then, inject 5 more players with skill 2 at 6:05 pm. Then, inject 4 players with skill 4 at 6:09 pm.

Expected Output: Until 6:09:59 pm, no match should be formed, and Q_1, Q_2 should have 6, 4 players enqueued respectively. Then, at 6:10:00 pm, one match should be formed consisting of all the 10 injected players. Each team should have 3 players of skill 2 and 2 players of skill 4. Afterwards, all queues should be empty.

Explanation: Since at 6:10 pm, the elapsed wait time of player 1 (the first enqueued player who got injected at 6:00 pm) exceeds the queue threshold value (10 minutes), it prompts cross-tier matchmaking, which results in the following skill makeup: 2-2-2-4-4 and 2-2-2-4-4.

The experiments were run for an observation period of 3000 units. The following results were obtained with cross-tier matchmaking disabled:

- Median matches created: 7
- Average waiting time per player: 315 units
- Fraction of players who quit: 17%
- Fraction of players who got rejected: 19%

The following results were obtained with cross-tier matchmaking enabled:

- Median matches created: 7
- Average waiting time per player: 306 units
- Fraction of players who quit: 14%
- Fraction of players who got rejected: 16%

It seems that cross-tier matchmaking helps decrease the average waiting time and the number of players who quit or got rejected. However, the difference seems to be very small. Further study should be done to explore these dynamics in deeper detail and get results with more confidence.

V. Simulation Results and Inference

Fig. 5 displays the model's state between the times ($1880ms - 1990ms$). Several things are happening here. At the time of Player ID - 76 (at $1896ms$), the phase of the queue

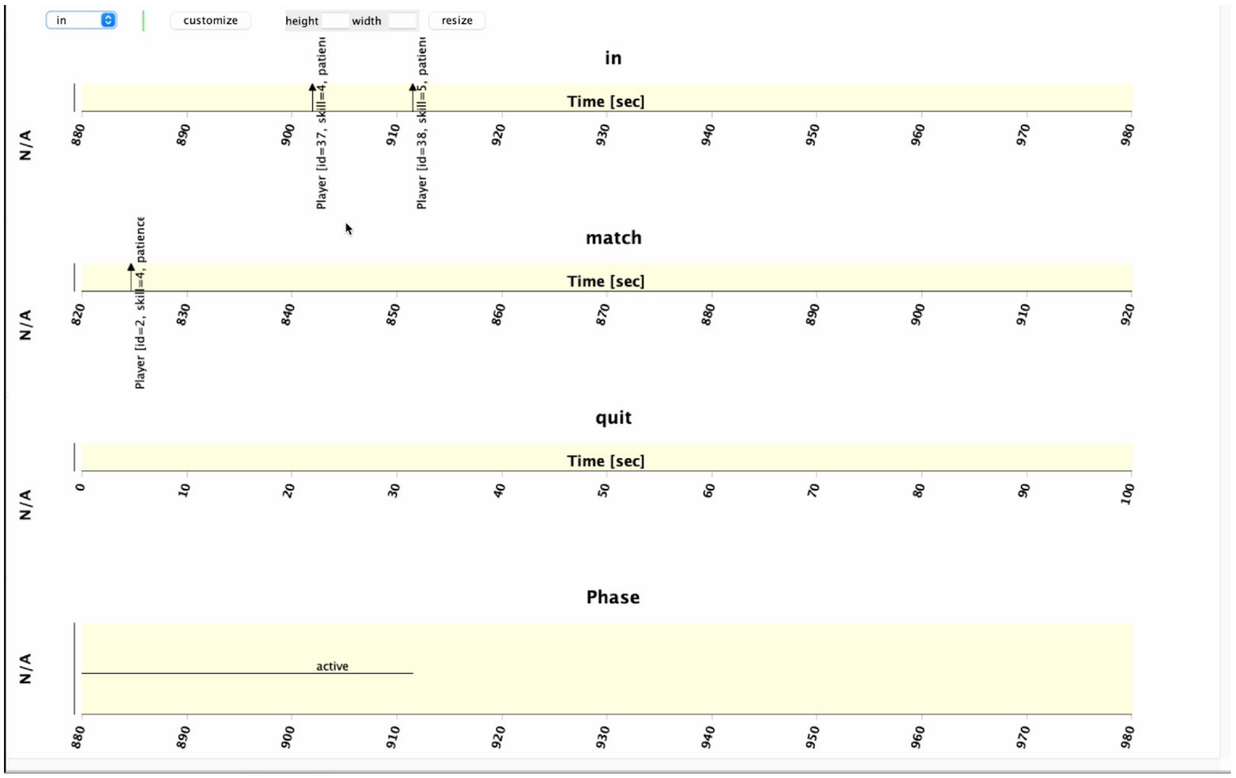
in
customize
height
width
resize
in
Time [sec]
N/A
Player [id=37, skill=4, patienc
Player [id=38, skill=5, patienc
match
Player [id=2, skill=4, patience
quit
Phase
active

Fig. 6. Arrival of Player ID 37

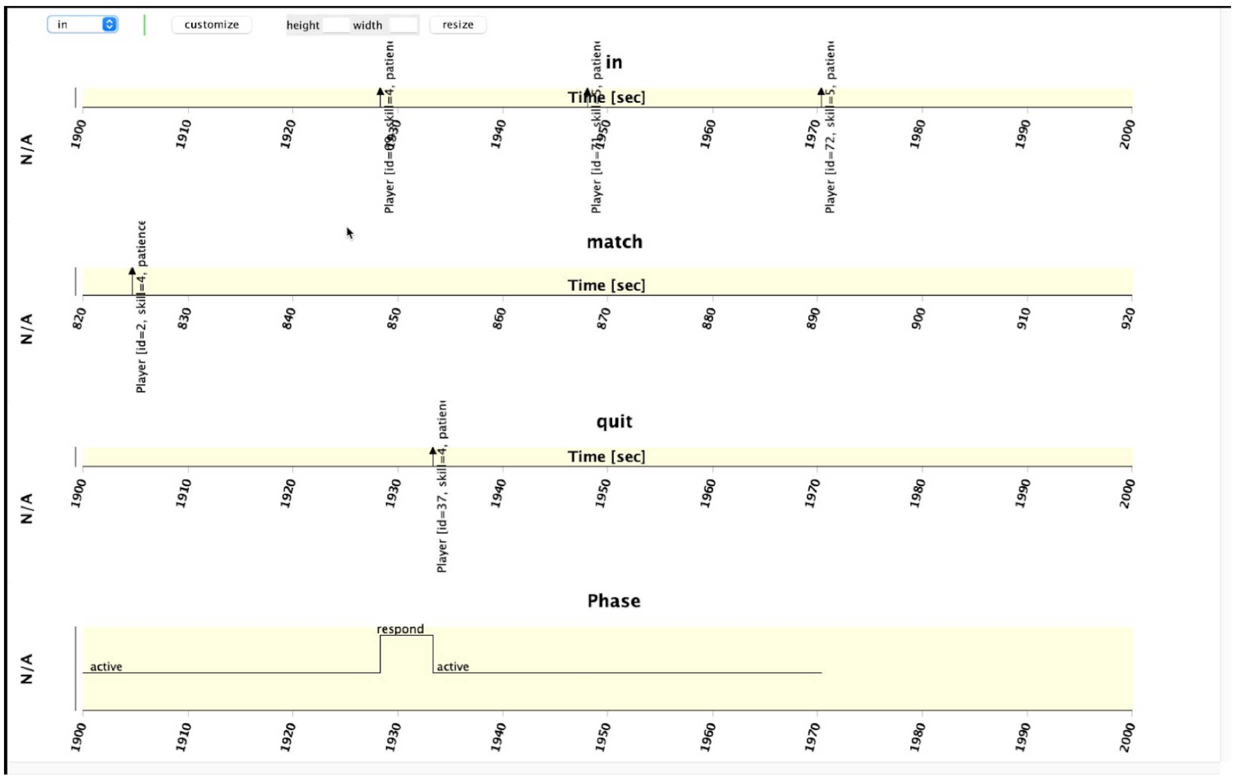
in
customize
height
width
resize
in
Time [sec]
N/A
match
Player [id=2, skill=4, patience
quit
Player [id=37, skill=4, patienc
Player [id=72, skill=5, patienc
Phase
active
respond
active

Fig. 7. Quitting of Player ID 37 and phase change

changes from “active” to “respond” this occurs due to the matching condition being satisfied and thus creating a match (which is evident at $t = 1892ms$), and the phase are reverted to "active" to receive more players. Another observation that can be made from Fig. 5 is at $1815ms$ when Player ID - 42 quits the match due to the patience threshold being crossed.

To elaborate more on the quitting process, Player ID 37 arrives at the input port at t = 902ms and waits. This state is displayed in Fig 6. Player ID 37 waits until 1934ms and finally decides to quit as his patience threshold (900ms) has been exceeded, and this information is relayed to the Activity Logger in the "respond" phase (as seen in Fig ¿¿+1). These successive states are displayed in Fig 6 and Fig 7.

VI. CONCLUSIONS

Throughout the entire modelling and simulation of this problem, we aimed to match 10 skilled players of similar skills and observe the difference in the relative waiting times that were achieved based on the matchmaking algorithm parameters.

The strategy was always to match players within the same tier before the cross-tier wait-time threshold ran out. The observations conclude that cross-tier matchmaking does indeed help decrease the average waiting time per player. Moreover, after assessing the different emulated scenarios, we can say that the relative wait times are more efficient when cross-tier matchmaking is enabled, as the fraction of players who quit due to waiting too long went down. To conclude, cross-tier matchmaking can be an excellent technique to make the queues more efficient and improve the player waiting times, albeit at the cost of higher skill disparity in a fraction of the game matches. Further analysis and more creative exploration of the algorithm have potential.

VII. Contributions and Skills Acquired

This was a two-member project by Aneesh Ahmed and myself. I was mainly responsible for identifying the initial problem scenario and devising an abstract solution to model it. I also contributed to the greedy algorithm/matchmaking logic used in the model. Programmed the logic for the Activity Logger that processes all the data going in and out of the high-level model to produce statistics such as matched percentage, unmatched percentage, number of players whose patience threshold has been crossed, mean and median of weight times etc.
Throughout the project, I gained a fresh perspective on Object-oriented modeling with the addition of simulation and solidified previously known concepts such as hierarchy, abstraction etc. Gained a good understanding of using the JAVA simulator framework (DEVS).

VIII. REFERENCES

[1] Andrej Hadji-Vasilev. (2022, Aug. 08). 23 Video Game and Online Gaming Statistics, Facts & Trends for 2022 [Online]. Available: https://www.cloudwards.net/online-gaming-statistics/

[2] M. Xu, Y. Yu, and C. Wu. (2019, Nov.). "Rule Designs for Optimal Online Game Matchmaking." *Arxiv* [Online]. Available: https://mingkuan.taichi.graphics/publication/2021-matchmaking/matchmaking.pdf

[3] B. P. Zeigler, H. S. Sarjoughian, Sunwoo Park, J. J. Nutaro, J. S. Lee and Y. K. Cho. "DEVS modeling and simulation: a new layer of middleware" *Proceedings Third Annual International Workshop on Active Middleware Services* [Online]. 2001, pp. 22-31. Available: https://ieeexplore.ieee.org/document/993717

[4] Kim, Sungung, Hessam S. Sarjoughian, and Vignesh Elamvazhuthi. "DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring" *Proceedings of the 2009 Spring Simulation Multi-conference* [Online]. 2009. Available: https://dl.acm.org/doi/10.5555/1639809.1655390