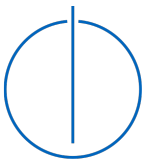
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Game Engineering Principles: Fair Gamespaces for Competitive Multiplayer Games

Felix Bartossek



DEPARTMENT OF INFORMATICS

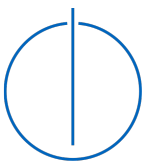
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Game Engineering Principles: Fair Gamespaces for Competitive Multiplayer Games

Game Engineering Prinzipien: Faire Gamespaces für kompetitive Multiplayer-Spiele

Author:	Felix Bartossek
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Daniel Dyrda, M.Sc.
Submission Date:	2022-02-11



I confirm that this Bachelor's Thesis in Informatics: Games Engineering is my own work and I have documented all sources and material used.

Munich, 2022-02-11 Felix Bartossek

Abstract

Good competitive multiplayer games need to be fair. As fairness is a multifaceted topic, it is interesting to approach subsections of the whole game and to find aspects that make the game fair. This work approaches the gamespace specifically, on the search for general principles of fair gamespaces, applicable to a broad range of games.
Furthermore, this work proposes a formalization, which allows developers to analyze gamespaces at any point of the development process, and to check whether the proposed principles are fulfilled. To put this way of analysis to the test, we apply it to the Summoner's Rift, the most relevant gamespace of the successful Multiplayer Online Battle Arena "League of Legends".

Contents

1 Introduction

There are many reasons to play a game. Some players like the challenge, others want to relax or to simply escape from their day-to-day-lives. From arcade-high-scores to the increasing popularity of e-sports, competition has always been an incredibly important factor: players want to measure their abilities, and show the others who's the best. This means, that one of the most important aspect of such competitive games is fairness. As opposed to gambling, in competitive games there is no point in being the best, when the own score is simply a result of chance instead of skill, or if the game is obviously rigged in the own favor - but there also isn't much incentive to try, if the opponents have an unfair advantage, and the own victory seems impossible.
Therefore, almost all successful competitive game should aim to be fair, to remain successful over an extended time [9]. Fairness is a basic, shared value and a core of playing: An unfair game tends to be less enjoyable for the party at a disadvantage, ultimately making them lose interest in the game as a whole [14]. There are many different attempts to define fairness. The classical measure for fairness in games is often stated as: "a game is a fair game if its game-theoretic value is a draw and both players have roughly an equal probability on making a mistake" [9, p. 17]. Another common definition for a fair game is, that the winning ratio for all teams is equal [17]. These simple definitions become less and less useful, the more complex a game becomes: the more complicated and intricate a game becomes, the more vectors of fairness can be considered - while the fairness of a game of "Rock, Paper, Scissors" is decently easy to measure and describe, the fairness of more advanced games, especially digital games, remains harder to grasp as a whole. Especially with asymmetric games, the keeping of balance and therefore fairness becomes harder. It can no longer simply be achieved by giving each player the same means to achieve their goal, instead it it necessary to balance the values of each players possibilities against each other.
To create and maintain a truly fair game means to be aware of all the possible vectors of unfairness, and to balance them. In a video game, this might be the playable characters, the task, the gamespace and many more things. Creating guidelines for achieving this kind of overall fairness could be a way to make game engineering simpler.
A gamespace is commonly defined as "the virtual space in which a video game takes place" [16]. This includes, what may be called a "map" or a "game board", but contains additional information about the game, instead of just the pure spatial architecture, like

the locations of resources.
This part of a game is always interesting: It is the playground in which the actual gameplay takes place, the environment to which all the other rules and limitations of the game correspond. Having a thoroughly balanced and fair gamespace can be a solid base for an overall fair game. Therefore, having accompanying guidelines for the engineering process of a gamespace can help to reduce unexpected sources of unfairness in the game.
The traditional rule of thumb for fair gamespaces is to keep them symmetric [12]. This relies on the assumption that a symmetric gamespace presents both teams with exactly equal opportunities, which can be problematic: neither must a fair gamespace be symmetric [14], nor must a game with a symmetric gamespace be fair (as will be shown in 6). This assumption should therefore not be the sole deciding factor.
As fairness becomes increasingly difficult to grasp and define the more complicated a game gets, it can be useful to achieve fairness in select sub-parts of a game, and to continue from there. Providing a fair gamespace is an important step towards the goal of an overall fair game.
This work will focus on providing guidelines and principles specifically for the fairness of gamespaces of competitive multiplayer games.

1.1 Related Work

Analyzing games is a broad field. The specific analysis of gamespaces is a tiny niche among this bigger cosmos. As the game, that this work will use as an example, is part of the MOBA[1] genre, analyses of other MOBA games or games of adjacent genres are of special interest.
Reddad et al. [12] focus less on fairness (as they follow the common assumption that balance is simply achieved by symmetry), but propose an interesting way of determining the quality of a map for the RTS-game[2] StarCraft 2. Their algorithmic approach is one that could easily be expanded or adapted to different games.
Wu et al. [17] take a broader approach to the MOBA genre as a whole, and try to find distinct aspects of fairness in different games. Their approach strongly relies on data from professional tournaments, which potentially doesn't coincide with the experiences of most players, but can nonetheless be expanded to have a broader perspective.

[1] Multiplayer Online Battle Arena, a popular genre of competitive online multiplayer game; the gameplay and structure described in 3.1 are typical for the genre
[2] Real Time Strategy, the genre from which MOBAs developed [7]

2 Process

2.1 General Approach

To analyze entirely different gamespaces, it is necessary to have a common abstraction for all of them. The visible surface, the aesthetics, are mostly irrelevant to this analysis. We need the functional gamespace [14]: the topology of the gamespace (rather than the exact geometry and graphics), enriched with data about resources and the importance of certain areas. Therefore, this paper will first propose a way of formalizing gamespaces, and then establish common principles of fairness, that can be measured and checked within the formalization. While it would certainly be possible to automatize the creation of such a model, creating the model by hand, with insights into the actual game, is likely to give better, more accurate results. This can be done along with the creation of the gamespace, to immediately incorporate the same creative thoughts that led to the inception of certain parts of the gamespace into the model as well. Finally, to create a first such formalization, a pivot-gamespace was selected, that was already fair in many aspects, and could be tested on.
The goal of this process is to have a formalization that develops along with the gamespace. Changes in the structure of the gamespace are reflected in the model, and checked for consistency with the proposed principles. This does not only work for the first construction of the gamespace, but can be especially interesting during the process of extending and adjusting an existing gamespace, as it prevents new decisions from destroying an established balance.

2.2 Structure of the Formalization

The functional gamespace mainly represents the topology of the actual gamespace, preserving distances and connections between distinct points within the space. As such, a graph is the obvious choice for a model [14]. Nodes of the graph represent locations within the gamespace, that are interconnected and can be navigated through. As the gamespace of most competitive multiplayer games is continuous, this leaves further questions - how big is a node, how are distances between nodes treated, how is it assured that every location within the gamespace is clearly assigned to a node?

There are a few aspects such a graph needs to accommodate: The structure of the graph should allow for every point within the gamespace to be clearly associated with one specific node. Nodes should be meaningful in the context of the game - there is no point in having several nodes that have no distinguishing factors at all. Finally, nodes should be clearly connected to or disjoint from other nodes, without ambiguous cases. For these reasons the model uses an approach, that is similar to a Voronoi partition [1]: nodes are defined by a center point and their neighbours, creating borders exactly between the centers. Therefore, any given point within the gamespace belongs to the node to which center it is closest. The node centers should be placed carefully and by somebody with deep understanding of the game in its entirety and the gamespace especially, ideally a designer, as the placement of the nodes can indeed vastly improve the overall quality of the model.
Connecting these nodes to a graph can be handled in different ways. Simply having labeled edges that describe the distance between the centers of two neighbored nodes would certainly make the construction process simpler, but lead to great inaccuracies, depending on the direction of transition through a node. Instead, nodes contain a lookup-table of connections via this node. This way, there is no single distance through a node, but a set of distances depending on the direction. Constructing this lookup-table for every node is an essential part of the formalization, and ensures a better result for the measuring of distances between nodes.
This is visualized in figure 2.2: Storing the direct connections between A and B, B and

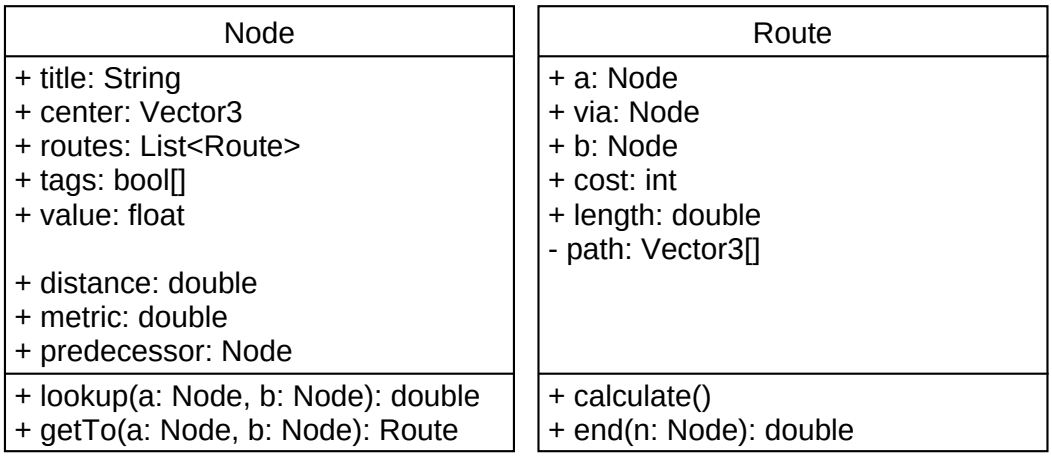


Figure 2.1: Classes for the representation of a node and a route

C and B and D would result in long, inaccurate distances when looking up a path (e.g. A-B-C). If, instead, the paths across node B are stored (visualized as magenta lines), looking up the paths will result in more accurate distances. On a bigger scale, this means that every node contributes only the cost of traversing this node in the given direction (i.e. the part of the stored path that lies within the blue Voronoi-borders) to the cost of a longer path (as visualized in figure 2.3). This results in paths that present a lower limit for the actual path, as each route through a node takes the shortest possible way through the node in the desired direction. This leads to what appears as "jumps" in the longer path: the routes through neighbouring nodes don't necessarily connect, as each route is calculated independently, with the goal of taking the shortest way for this part of the path.

The way these structures are represented can be seen in figure 2.1. The representation of a node contains the described components, plus additional variables that are used for pathfinding. Routes provide both the path through the respective node, as well as the ends of the route in the adjacent nodes, which are relevant for the beginning and end of a longer path. For further details, see chapter 5.

2.3 Measurements

Once a model is established and set up, measuring various distances and connections is simple. The formalization makes use of a modified version of the A* algorithm [10], that takes into account the special representation of distances. Once set up for a gamespace during development, desired criteria and principles can be set up as test-cases, that are automatically recomputed as the gamespace is changed. This is further supported by added tags in the nodes, to mark special locations like buffs and objectives. This gives the model further options to accurately measure aspects that indicate fairness.

2.4 Choosing a Game

To work on a suitable representation, it was helpful to use a initial gamespace as a base. In order to further ease the process, this gamespace needed to be well-tested, already relatively fair and not too overly complex. Also, it needed to be from a game of which the author of this paper had appropriate game knowledge, to fully understand the context for the gamespace.

Given these criteria, the Summoner's Rift from League of Legends was a fitting choice. It has been in use (although with many modifications over time) for well over a decade now, and as League is among the most-played online multiplayer games, it has been extensively tested and refined, with a lot of attention to the balance. This does not

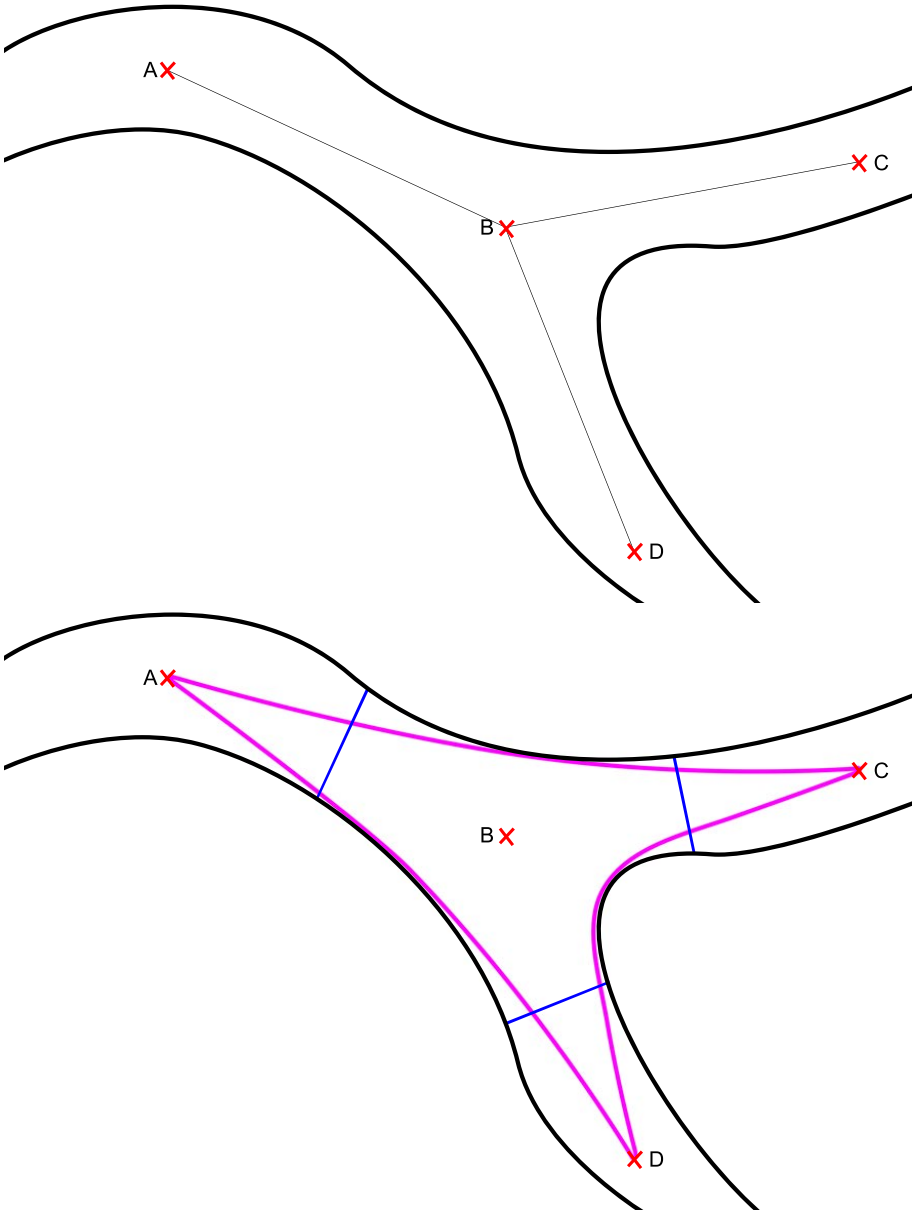


Figure 2.2: Three nodes (red crosses mark the centers, blue lines mark the borders, magenta lines mark the paths stored in node B, grey lines mark direct paths between node-pairs) around a narrow crossing

mean the Summoner's Rift is perfectly fair - but it is fair enough to be a suitable base for this project.

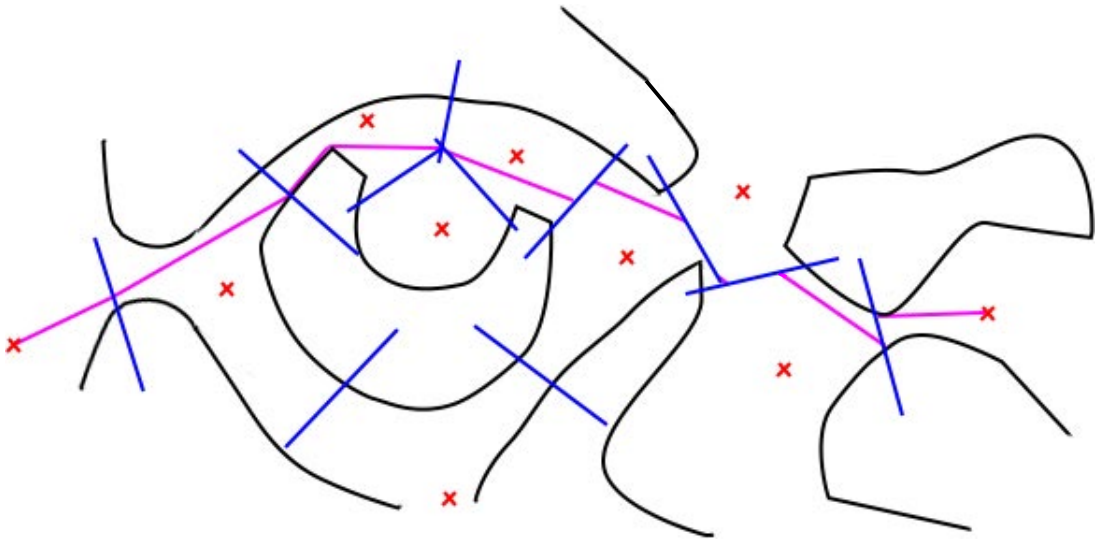


Figure 2.3: A longer path: Only the contributions of each node are shown, the route to traverse the respective node from its predecessor to its successor. Simplified example from the Summoner's Rift (see section 3.1)

3 League of Legends

League of Legends (LoL) is a MOBA developed by Riot Games, released in 2009. Inspired by the success of DotA[1], LoL pioneered the "game as a service"-approach, and is continuously being extended, improved and balanced [2]. LoL is free to play, earning its revenue via the in-game sale of cosmetics. More than a decade after its initial release, the game has secured its place as one of the most played online multiplayer games, and the second-most watched e-sports game in the world [3]. Due to the great strategic depth and diverse gameplay, LoL is rather difficult to pick up, but once a player gets a grasp of the game, they often keep playing over a long time [2].

3.1 Structure of the Game

The following sections are based on the official how-to-play of LoL [15] and the personal experience of the author of this paper.

3.1.1 The Summoner's Rift

While League of Legends has various different game modes and according gamespaces, there is only one with competitive relevance, which will be the focus of this analysis: the Summoner's Rift. On the Summoner's Rift, two teams of five players (the blue team, which has their base in the lower left corner of the Rift, and the red team which has their base in the upper right corner of the Rift) compete against each other to conquer and destroy the opposing team's nexus. The Rift consists of the two bases, connected by three lanes (toplane, midlane, botlane), and the jungle between them.
A base contains the fountain, the nexus, three inhibitors and five turrets.
The fountain is the spawn-point[2] of the players. This is the only place where players can buy items to upgrade their stats, and is only admissible for players of the own team - entering the enemy's fountain means to take enormous damage, to prevent spawnkilling[3]. Standing in the own fountain allows the player to rapidly regenerate

[1] short for "Defense of the Ancients", a popular Warcraft 3 mod that largely shaped the MOBA genre [7]
[2] The location in which players first appear in the beginning of a game, and reappear after they died.
[3] the act of killing a player immediately after they respawn, preventing them from taking any defensive actions

Figure 3.1: The minimap of LoL, as it appears to a player on the blue team.

health and other player-specific resources like mana.
The nexus is a massive structure that presents the main objective of the game. The destruction of a nexus immediately ends the game, marking the team that did it as the winners. On the minimap (Figure 3.1), the nexus is represented as a circle with four rectangular protrusions and a tiny square in the middle, colored in the respective team's color.
Inhibitors are smaller structures, resembling the nexus. Destroying an inhibitor is essential to progress further into the enemy's base, and allows the own team to spawn super minions. On the minimap (Figure 3.1), the inhibitors are represented as circles with a tiny rectangle in the middle, colored in the respective team's color.
Turrets (also called towers) are structures that defend a certain area around them. Each turret shoots beams of energy that deal increasing damage with the amount of consecutive shots. Turrets will attack any enemy unit that enters their range, but only one at a time. An enemy champion, who attacks an allied champion within the range of a tower will immediately get focused by the tower, until they leave its range. To attack any enemy structure (i.e. turrets, inhibitors, the nexus), the structures before it

need to be destroyed[4].
Between the bases lie the lanes. Each lane has a neutral middle ground, the rest of the lanes is protected by turrets (two turrets per lane per team). In each base spawn minions (NPC[5] entities that align with a team, push the lane[6], and can be killed by opposing players for gold and experience). There are different kinds of minions (melee-, caster-, cannon-, and super-minions) that appear in different quantities and intervals, and deal different amounts of damage, have different amounts of HP and are worth different amounts of gold and experience.
Between the lanes lies the jungle, a landscape of twisted, narrow paths and occasional clearings (called camps) where neutral monsters can be slain for gold, experience, and in some cases either individual or team-buffs. On the minimap (Figure 3.1), buff-camps are represented by a tiny, orange animal-head. Regular camps that don't give buffs are represented by orange diamonds. The most important of these objectives are the Dragon ("drake") and Baron Nashor ("baron"). Both spawn in the pits in the river, the baron between top- and midlane, the drake between bot- and midlane. Both have unique buffs for the team that claims them, and develop over the course of the game:
The Baron only spawns after 20 minutes, before that, his pit is occupied by the rift herald, a neutral objective that allows the player who claims it to greatly damage a turret. After that, the Baron spawns. It gives all living members of the team that claims it a buff that empowers minions in their proximity and gives them a quicker recall[7]. On the minimap (Figure 3.1), the baron is represented by a purple icon.
Each Drake, on the other hand, has an element. The elemental drakes give the team that claims them a unique buff, depending on said element. The first three drakes are of different elements, afterwards there are only drakes of the same element as the third one, and the entire Summoner's Rift changes according to that element. Once one team has claimed four drakes, they gain an additional buff (the drake soul), and no more elemental drakes spawn - instead, there are only Elder Dragons from now on, who give the team the temporary passive ability to execute low-health enemies. The dragon is the most important jungle objective in the current meta, to a point where gaining the

[4]for example: To attack the nexus, the two turrets in front of it (the nexus-turrets) need to be destroyed. To attack the nexus-turrets, at least one inhibitor needs to be destroyed. To attack an inhibitor, the turret in front of it (the inhibitor-turret) needs to be destroyed. To attack an inhibitor-turret, the lane turret in front of it (the tier 2 - turret) needs to be destroyed. Finally, to attack a tier2-turret, the tier1-turret on the same lane (the turret closest to the neutral ground) needs to be destroyed.
[5]Non-Player Character, any character in a game that is not directly controlled by a player
[6]"pushing a lane" means to advance on the lane from the own base, attacking and killing enemy units on the way, and sieging, damaging and eventually destroying enemy structures on the lane.
[7]Players can recall at any point in the game. This teleports them back into the fountain of their team. Recalls are interrupted by moving or taking damage during the channeling period, which usually takes 8 seconds. The baron-buff halves this duration.

drake soul as a team can mean an almost guaranteed win[8]. On the minimap (Figure 3.1), the drake is represented by a respective elemental icon, in the depicted case an orange flame for the infernal drake.
The Rift is centrally symmetric, with only one notable exception: the herald/baron takes the place of the drake on the other side.

3.1.2 Champions

Before the beginning of a game, each player picks a champion to play as. The amount of playable champions is steadily increasing - when LoL was first released, there were only 40 individual champions [2], and by now there are more than 150. Each champion has unique abilities and stats, with many different playstyles and possible combos. Each champion can only be selected once per team, and in most game-modes even only once per game. Champions cannot be changed later in the game.
Killing enemy or neutral units or structures allows players to earn gold and experience. Gold is spent in the shop, to purchase items that enhance the stats of the played champions, or give entirely new passive effects. Experience grants the champions level-ups, which allow the player to improve their champion's abilities. The maximal level is 18, at which all abilities reach their maximal strength.
As the teams consist of five players each, the players have to arrange themselves on a map with fewer lanes than players. Commonly, the lane-assignment is as follows:
One player pushes the toplane. The toplaner usually relies on a solitary playstyle, putting high effort into farming[9] and scaling[10]. Common champion-choices for the toplane are tanks[11] and bruisers[12]. As toplaner, encounters with other players are rare in the early stages of the game.
One player pushes the midlane. As the midlane lies in the diagonal of the Rift, connecting the bases in a straight line, it is the shortest lane. This means, that minions will reach the neutral ground of the lane faster than anywhere else, allowing for quicker wave-clear[13]. Midlaners commonly employ an aggressive playstyle, choosing assassin-[14]

[8] At this moment, teams that claim the weakest possible soul have a winrate of more than 80%, according to `https://www.leagueofgraphs.com/rankings/drakes` on 2022-19-01
[9] killing minions to gain gold and experience
[10] farming as efficiently as possible during a weaker phase to become strong through better items
[11] defensive champions that often do not deal a lot of damage, but have strong resistances and/or regenerative abilities, and strive to immobilize enemies to set up opportunities for the own team
[12] champions that deal a lot of damage at close range and can easily sustain and regenerate a lot of damage
[13] Minions spawn in waves, with fix delays between each wave. Clearing a wave means to eliminate all minions of this wave.
[14] mobile champions that can deal lots of damage in very little time, but can easily be killed themselves

or mage-[15] champions. The midlane also allows the player to easily roam[16] to other lanes, helping out players with objectives or in teamfights.
The botlane is usually pushed by two players: while one of them is the primary botlaner, the other one takes up a supportive role. In most games the botlaner is a strong but fragile carry[17], often a marksman[18] or mage. The support, on the other hand, usually only gains gold passively, and therefore operates on a scarcer budget. They help their carry in different ways - as a tank, an enchanter[19] or a mage - in the early game, and roam later to support the entire team.
The final player takes the role of the jungler. They farm gold and experience from the various neutral monsters in the jungle. The jungler provides their team with vision, takes bigger, neutral objectives that buff the entire team, and ganks (ganking a lane means to visit said lane as a jungler, assisting the allies and attacking the enemies) lanes to put the respective laners at an advantage. There is a wide variety of possible viable champions for the jungle, but most of them are usually very mobile (e.g. possessing abilities that allow them to traverse walls), to quickly get from lane to lane.

3.1.3 Phases of a Game

In the beginning of a game, all players spawn in their bases, and buy their starting items from the shop. They position themselves at crucial positions in the jungle to either prevent an invade or to start an invade themselves[20]. The players then proceed to their assigned lanes or help the jungler with their first monster. All lanes are now being pushed by the opposing players, who may engage in short skirmishes, but mainly farm the minions. In this phase of the game there are few kills, usually only in case of a gank by the jungler. The first objectives get taken, and gradually even the first turrets. With the start of the midgame, the strict lane assignments are usually broken. There are more teamfights. more roams, objectives are often engaged by whole teams. The first dragons have been taken, and likely the herald too. In the lategame, the baron and the elder drake become the deciding objectives. The teams fight mostly together, trying to gain the deciding advantage that will win the game.

[15] champions that deal a lot of magic damage over some range, but are typically immobile and easy to kill
[16] quickly move from one lane to another, to assist allied champions or to score quick kills
[17] A strong champion that, through the assistance of its allies in the beginning becomes extremely strong later on.
[18] champions that deal a lot of physical damage over long distances, but usually don't have a lot of hp
[19] a supportive champion that helps their allies by giving them shields or healing, rather than damaging or immobilizing enemies
[20] invading means to enter the part of the jungle that is closer to the enemy team's base, to kill a buff monster or to surprise the enemy jungler

4 Principles

From the definition of the model and the knowledge of the game itself, it is possible to derive desired criteria and principles, and to assemble them in a form that allows a steady checkup during the engineering process. Setting up these principles as a kind of test-case allows the engineer to check at any point during development, whether the current iteration of the gamespace still fulfills the previously defined criteria. While some criteria might present principles that apply to nearly every game, others might be only desirable for certain types of games, or depend on game-specific variables. This chapter proposes a broad selection of principles that can be applied in this or a modified form to various different games. In the following, we will assume that there are two competing teams in the game. All of the principles are applicable to games with an arbitrary number of competitors, but they might need to be reworded.
While all of these principles are designed to support fairness and balance, there are two different kinds of principles that further different aspects of balance: The first kind attempts to create balance between the two opposing teams (4.1), and the second one keeps the balance between different players and roles (4.2), to prevent certain roles from overpowering everyone else in the game.

4.1 Horizontal Balance

We define horizontal balance as the balance between both teams. Neither team should be, from the beginning of the game, in a significantly advantageous position [14]. The simplified map from figure 4.1 will serve as an example for some of the following principles. Cyan lines represent defense-lines for team B, magenta lines represent defense-lines for team A. The labeled circles represent the bases for the teams, the smaller circles represent resources.
We propose the following principles with regard to horizontal balance:

4.1.1 Resource Access

The access to resources within the game should be equal. This does not have to be perfectly symmetrical - while symmetry certainly makes balance easier to ensure, it is

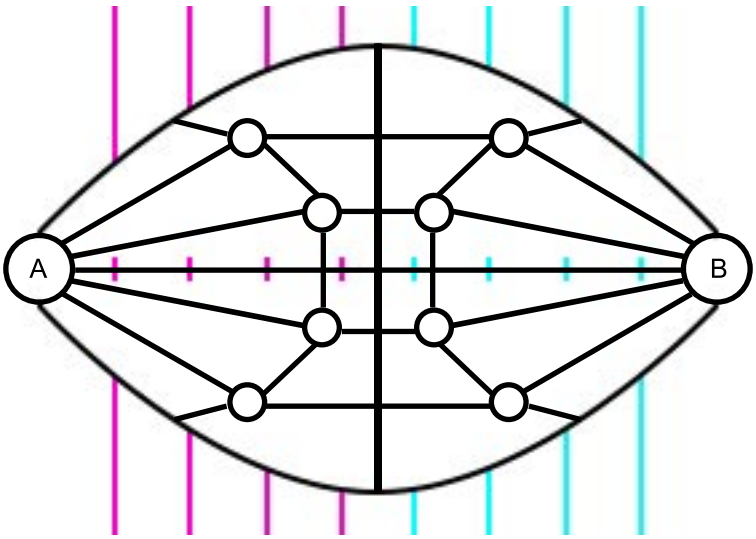


Figure 4.1: This simplified map of an arbitrary MOBA is symmetric. It will serve as an example for some of the following principles.

not the only way to achieve it [14]. Resources appear in various forms across different games. They are goods available to players, usually after completing a task (like killing an NPC or claiming a control point), and often without necessarily interacting with an enemy player. These may include, but are not limited to: experience and gold (often gained by slaying neutral or enemy NPCs, e.g. creeps in DotA 2 [4]), healing (often provided in fixed locations, e.g. health packs in "Overwatch" [6]) or buffs (often gained by slaying a particularly tough NPC, e.g. the drake in LoL, see 3.1).

Every team should have equal opportunities to access resources. These resources may be distributed asymmetrically, of course. To properly measure resource access, resources - or more precisely the locations of said resources - can be marked with a tag for quicker analysis and a relative value of the resource. Said values can and should be changed throughout the testing process, depending on the resource's impact on the actual gameplay [14]. Now, the distances from bases to resources for each combination of team and resource can be measured. We propose an access rating a for both teams as a suitable measurement:

$$a(t) = \sum_{r \in resources} \frac{value(r)}{distance(r,t)}$$

Checking, whether this value is equal[1] for both teams is trivial.

[1] or equivalent with a chosen margin; depending on the placement of the nodes the model can bring a

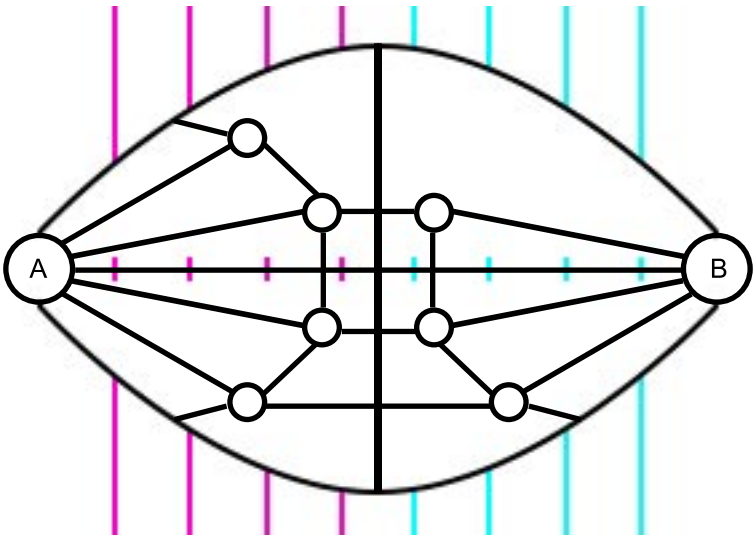


Figure 4.2: Given equal values for all resources, team A has a better access rating.

To push this facet of fairness further: **Every team should have equal opportunities to claim multiple resources one after another.** Depending on the structure of the gamespace, both teams can have equally long distances to various resources from their bases, but the distance between the resources can produce great imbalance (figure 4.3). Measuring this requires further calculations: it is necessary to accumulate the shortest paths that connect multiple resources for both teams.

Another approach to fairness in regard to resource access is the priority on a resource the teams have. A team has higher priority on a resource, if the path from this team's base to the resource is shorter than the other team's. Again, the value of a resource should be took into consideration: **Both teams should have priority on resources of equal value.**

The availability of resources can be examined in greater detail. Most resources are not available permanently, but often have a cooldown, and some resources require a certain time to claim them; this can be included in the analysis of their availability.

4.1.2 Paths

Within a larger gamespace, there are more locations of interest than just resources. Neutral spaces will be the most common points of conflict in the early stages of a game,

certain deviation with it

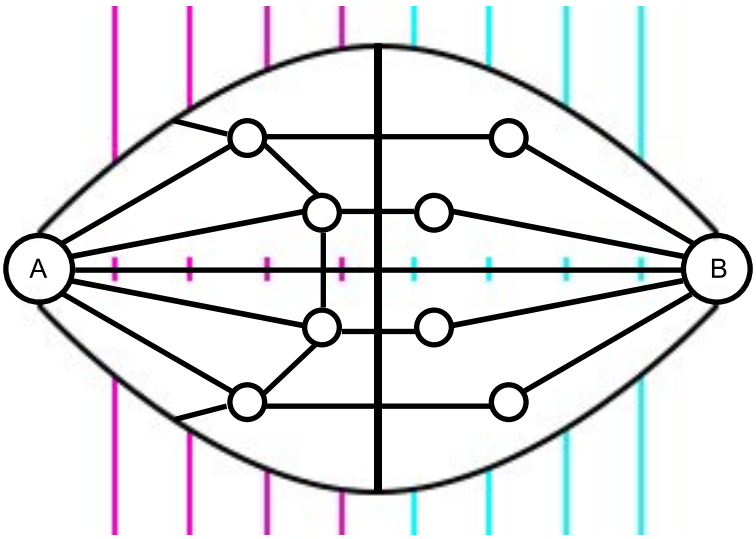


Figure 4.3: Given equal values for all resources, both teams have an equal access rating, and yet team A is at a great advantage to accumulate multiple resources in limited time.

and their locations are important. Detecting crossings[2] and bottlenecks[3] can greatly help the understanding of the gamespace.

Neutral territory should, in the beginning of the game, be equally far away from all teams' bases. In gamespaces without defensive entities (like turrets in LoL, see 3.1), this will often be established naturally, as the middle[4] between the bases is the first point where players of both teams will collide, if they start moving out at the same time. In gamespaces with specifically established defensive lines, this is of greater importance: the team with the shorter way to the neutral space will have a defensive advantage for the first encounter, but after that they can be pushed back to their base more quickly. Neutral spaces can be measured easily - defensive lines can be marked on the nodes as a tag, and the space between the last defensive line of one team, and the first of another is neutral space.

The most common crossroads should lie in neutral territory. If the most central, most frequently crossed point of a gamespace lies within the space inherently dominated

[2]Places in the gamespace in which many paths intersect
[3]Places in the gamespace that are frequently used but are narrow and allow for little evasive maneuvers
[4]Not necessarily the actual middle of the gamespace, but the point to which the traveling distance is equally far from both bases.

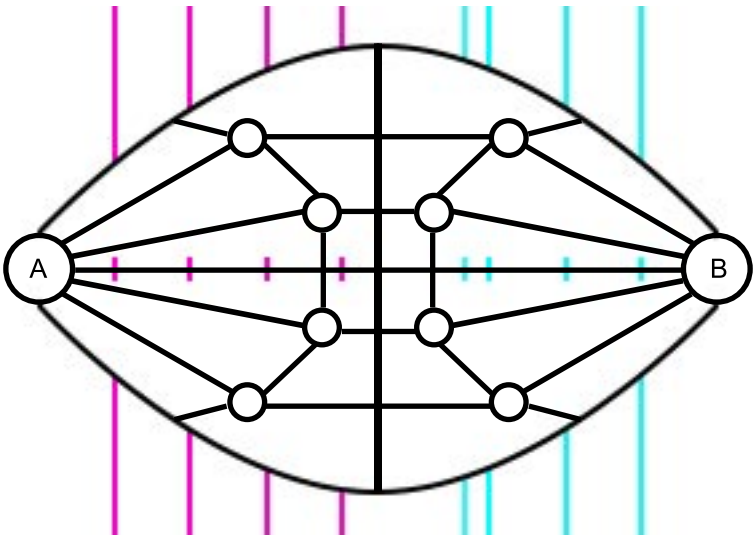


Figure 4.4: The neutral area in this example is closer to the base of team B, putting them at a disadvantage.

by one team, the balance of the gamespace is certainly disturbed. Finding important crossings is not complicated, but possibly expensive: By calculating all possible paths (and, for better accuracy, optionally weighing them with the value of their end nodes), and counting the appearances of each node, the most important crossings can be found.

Furthermore, **no base should be locked behind bottlenecks.** Having a base that is only accessible through bottlenecks makes it easy to trap all players of a team in their base[5]. This corresponds to a second principle: **Spawntrapping[6] should not be possible.** Doing so would make the game effectively unplayable for the trapped team, which enables the other team to harass them indefinitely. This behaviour is often considered bad manners (behavior that serves no purpose other than ruining the fun for other players), and should be prevented entirely.

Another important part of many gamespaces are paths that connect points of confrontation (e.g. neutral spaces) with other points of confrontation and objectives. These paths are often used to create situations in which one team gains numerical superiority to surprise their opponents (often called "gank", see chapter 3). While these paths

[5]There are exceptions to this, although they are rare, e.g. a hypothetical base that is only accessible through bottlenecks, but in a star-like manner with more exits than players, making it nevertheless impossible to lock down.

[6]Trapping players at their spawnpoint, often in combination with spawnkilling

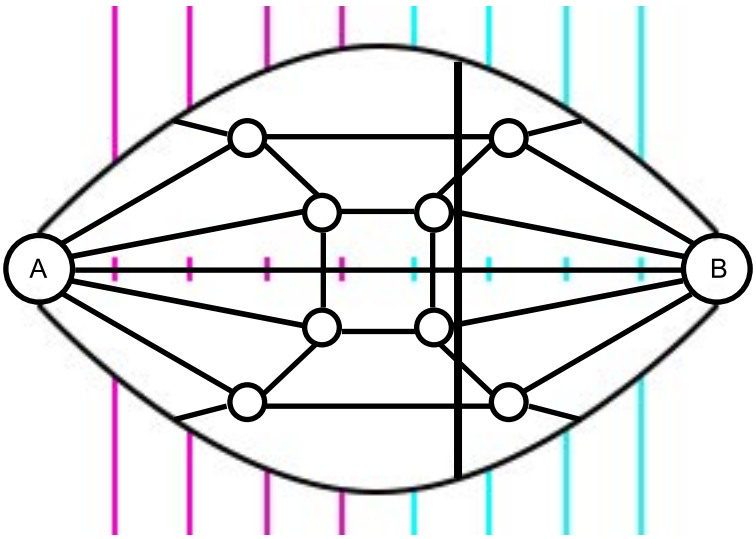


Figure 4.5: The most important crossing is moved closer to the base of team B, creating an imbalance.

are interesting for vertical balance (section 4.2) as well, **the length and availability of gank-paths should be balanced for all teams.** If players of one team have the possibility to freely move between two places of interest, while the players of the other team are constrained to their positions or need to take significantly longer paths, the more mobile team is at an advantage. This is also important for the amount of such paths: if one team has multiple different angles for a gank while the other team is limited to just one, the team with more options certainly has an advantage. To balance gank-paths, the places of interest can again be weighed by their importance (just like resources can be balanced by their values): Teams might be at an advantage for different places of interest, as long as their values even out.

Another important factor in regard to gank-paths are the objectives along the path. If one team has the possibility to easily take an objective along the way of a frequent gank path, while the other team can't, a player from the first team gains more utility from taking that path. **Therefore, the access to objectives along common gank-path should be balanced.** The objectives along a path can be easily measured by checking neighboring nodes for the relevant tags. This, again, should be weighed with the respective value of the resource.

4.2 Vertical Balance

We define vertical balance as the balance within a team, that allows players to feel like they have roughly the same level of impact on the game like their peers. In terms of the gamespace, this mainly means to keep the balance between players who stay in motion between objectives and combat situations and players who mainly stay in fixed spaces. We propose the following principles with regard to vertical balance:

As an important step towards this goal, **players should not be able to claim two resources at once.** This limits the speed at which mobile players proceed across the gamespace, and prevents excessive gain of resources in limited time. As resources usually have a set radius in which they can be claimed, this can be measured by making sure none of these radii intersect.

Another possible way to keep mobile players from having too much impact can be to set constraints for the length of gank-paths. **Gank-paths should have a set minimal length in relation to the path from the bases to the neutral space.** This is relevant only for the neutral space in the beginning of the game - as the game progresses and the neutral area shifts, this constraint may be broken. Finding a suitable relation between the two paths is task that relies on more than just the gamespace alone, since different games might have different requirements in this regard.

5 Algorithms

5.1 Setting Up the Routes

The routes, which are stored in each nodes' lookup-table, provide the basis for the entire model. These traversals always connect two nodes (a and b) via a third node in which the route is stored. The algorithm for calculating these routes works as follows (Algorithm 1):
At first, the exact path between node a and node b is calculated. This is done using the pathfinding algorithm of the game engine (Unity NavMesh etc). Here, the node placement is quite important: if the computed path doesn't actually cross the middle node, the node placement and connections aren't ideal and should be changed. Once the whole path is calculated, it needs to be trimmed. Having the entire path isn't very useful for later pathfinding - the relevant distance is the distance across the middle node, outside of the boundaries of the start- and end-node. The other parts of the route are only relevant for the beginning and end of a longer path, and can be stored separately.
Trimming the path is done as follows: At first, all the vertices of the path that lie within node a are culled. Then, the intersection point of the route and the border between the nodes is added. It will serve as the new starting point of the route. Now the vertices within the middle node are added, with another intersection point at the end, marking the end of the middle node. Once this route is assembled, its length can be easily calculated by summing up the distances between the vertices in order. The border between two nodes is the normal of the direct connecting line, in the exact middle between the node centers. In the presented case, the border function assumes both nodes to have the same y-coordinate. This may vary for different gamespaces and needs to be adapted for each case.

5.2 Finding Paths

Once the individual routes are set up, the model is ready to have various kinds of measurements run on it. Many of these require the model to provide pathfinding. To

efficiently find paths within the model, the A* algorithm [10] can be adapted to work on this kind of graph. This works largely as usual (Algorithm 2):
In the beginning, all nodes receive their starting parameters, which include an infinite distance, the added metric for quicker pathfinding (in this case, simply the euclidean distance between the center of the node and the center of the goal) and an empty predecessor. In the beginning, the start-node is evaluated. It is the only node that is evaluated differently from all others, as it has no predecessor - there can't yet be a route *through* it, but there has to be a route *from* it. In this case, the algorithm looks up the distances from the border of the start node to the centers of the adjacent nodes. In a second step it would be possible to collect routes via the adjacent nodes to the start node, which would require a different kind of route-lookup. The found nodes have their distance and predecessor set. Now, the star-node is removed from the unexplored set, and the usual algorithm begins.
The node with the shortest sum of distance and metric is chosen to be explored, and its neighbours have their distances set. The important change from the usual algorithm is, that the predecessor plays a role in the length of the paths. Therefore, each distance can only be set with the respective predecessor.
In the end, this algorithm returns the path rather than the distance, as it is relevant for various measurements. The distance is easily looked up from the path by adding up the costs of the individual routes of which the path is composed, and adding the respective ends of the first and last routes.

5.3 Measuring

To check the principles proposed in chapter 4, we need algorithms to measure the different properties of the gamespace. While some of these principles can be checked very generically, some require a stronger adaption to a specific game. This section will present some of the more broadly applicable algorithms, designed to work with the previous functions.
For the following algorithms we will need a set of nodes that contain resources, as well as the bases of both teams. These sets can be easily deduced from the set of all nodes by filtering it for the according tags. These sets can be computed outside of the measuring functions and stored for efficiency.
To calculate the access - ratings for a team (algorithm 3), the distances[1] from the respective base to all resources in the game are weighed with the value of the resources. This can be done for both teams, and the ratings can be compared (with a reasonable

[1] Actual travel-distances, calculated using the lookup and A* algorithms.

threshold).

Comparing the priorities both teams hold (algorithm 4) happens similarly. For each resource, the distance to both bases is measured. The team that holds priority on a resource gets the value of the resource added to their priority rating. The algorithm returns a tuple with the priority ratings for both teams. Should there be more than two teams, this algorithm needs to be adapted.

Algorithm 5 allows to calculate, how many routes between two sets of nodes cross every node. This has a multitude of applications, finding crossings between various subsets of nodes can help the engineering process, and one of these applications is finding the most important crossing(s) of the gamespace. To do that, the input for the function can either be two times the set of all nodes or, later on during the engineering process, with more knowledge of the game, two sets of all nodes, with some nodes appearing multiple times, weighed with their importance. The function returns a dictionary of all nodes, associated with the number of routes that cross them.

The resources along a path can be measured using algorithm 6. This can be used to evaluate gank-paths[2] for different teams. The algorithm collects resources in nodes that are traversed by the path and in nodes that lie adjacent to the path, and returns the set of resources. This can be evaluated for the amount or value of the resources. In this example it is assumed that the information whether a node contains a resource is stored in tags[0] of the node. Should this not be the case, or should there be multiple tags for resources, this needs to be adapted.

[2]These paths can either be calculated or obtained from gameplay tests.

Algorithm 1 Calculation algorithm for individual routes

```
1: function Calculate
2:     tempPath ← Pathfinding(this.a, this.b)
3:     this.path ← TrimPath(tempPath, this.a, this.via, this.b)
4:     this.length ← Pathlength(this.path)
5: end function
6:
7: function TrimPath(p, a, b)
8:     path ← []
9:     i ← 0
10:    while Dist(p[i], a) < Dist(via, a) ∧ Dist(p[i], a) < Dist(p[i], b) do
11:        i ← i + 1
12:    end while
13:    borderline ← Border(a, via)
14:    borderPoint ← LineIntersect(borderline, (p[i − 1], (p[i] − p[i − 1])))
15:    path.append(borderPoint)
16:    while Dist(p[i], b) < Dist(via, b) do
17:        path.Append(p[i])
18:        i ← i + 1
19:    end while
20:    borderline ← Border(b, via)
21:    borderPoint ← LineIntersect(borderline, (p[i − 1], (p[i] − p[i − 1])))
22:    path.Append(borderPoint)
23:    return path
24: end function
25:
26: function Pathlength(path)
27:    length ← 0
28:    for i ← 0, path.length − 1 do
29:        length ← length + Dist(path[i], path[i + 1])
30:    end for
31:    return length
32: end function
33:
34: function border(a, b)
35:    normal ← (−(b.center.z − a.center.z), a.center.y, b.center.x − a.center.x)
36:    normal.Normalize
37:    center ← ((a.center.x − b.center.x), a.center.y, (a.center.z − b.center.z)) × 0.5
38:    point ← (a.center + center) − normal
39:    return (point, normal)
40: end function
```

Algorithm 2 A* Algorithm, adapted to work with the node representation

```
1:  function AStar(start, goal, nodes)
2:      for all node in nodes do
3:          node.distance ← ∞
4:          node.predecessor ← null
5:          node.metric ← Dist(node, goal)          ▷ euclidean distance between centers
6:      end for
7:      start.distance ← 0
8:      for all route in start.routes do
9:          route.a.distance ← route.End(route.a)
10:         route.a.predecessor ← start
11:         route.b.distance ← route.End(route.b)
12:         route.b.predecessor ← start
13:     end for
14:     nodes ← nodes\start
15:     current ← null
16:     while nodes ≠ ∅ do
17:         current ← null
18:         for all node in nodes do
19:             if current = null then
20:                 current ← node
21:             else if next.distance + next.metric > node.distance + node.metric then
22:                 current ← node
23:             end if
24:         end for
25:         nodes ← nodes\current
26:         if current = goal then
27:             break
28:         end if
29:         for all route in current.routes do
30:             other ← null
31:             if route.a = current.predecessor then
32:                 other ← route.a
33:             else if route.b = current.predecessor then
34:                 other ← route.b
35:             else
36:                 continue
37:             end if
```

```
38:             if other.distance > current.distance + route.length then
39:                 other.distance ← ex.distance + route.length
40:                 other.predecessor ← current
41:             end if
42:         end for
43:     end while
44:     path ← []
45:     path.AddFront(current)
46:     while current.predecessor ≠ null do
47:         path.AddFront(current.predecessor)
48:         current ← current.predecessor
49:     end while
50:     return path
51: end function
52:   ▷ A* returns a path rather than the distance, which can be obtained via Lookup
53: function Lookup(path)
54:     l ← path.length
55:     if l = 1 then
56:         return 0
57:     else if l = 2 then
58:         directPath ← Pathfinding(path[0], path[1])
59:         return Pathlength(directPath)
60:     end if
61:     distance ← 0
62:     for i ← 0, l − 2 do
63:         step ← path[i + 1].Lookup(stops[i], stops[i + 2])
64:         if step > 0 then
65:             distance ← distance + step
66:         else
67:             return −1
68:               ▷ There is no connection between the nodes, this is an erroneous state!
69:         end if
70:     end for
71:     ▷ The result does not yet contain the distance within the start- and goal-nodes.
72:     distance ← distance + path[1].GetTo(path[0], path[2]).End(path[0])
73:     distance ← distance + path[l − 2].GetTo(path[l − 1], path[l − 3]).End(path[l −
   1])
74:     return distance
75: end function
```

Algorithm 3 Algorithm to look up the access rating for one team

```
1: function Access(base, resources)
2:     rating ← 0
3:     for all res in resources do
4:         rating ← (res.value ÷ Lookup(AStar(base, res))) + rating
5:     end for
6:     return rating
7: end function
```

Algorithm 4 Algorithm to assess the priority both teams hold

```
function Priority(base1, base2, resources)
    prio1 ← 0
    prio2 ← 0
    for all res in resources do
        d1 ← Lookup(AStar(base1, res))
        d2 ← Lookup(AStar(base2, res))
        if d1 > d2 then
            prio1 ← prio1 + res.value
        else if d2 > d1 then
            prio2 ← prio2 + res.value
        end if
    end for
    return (prio1, prio2)
end function
```

Algorithm 5 Algorithm to find important crossroads

```
function Crossroads(nodesA, nodesB)
    dictionary ← {}
    for all a in nodesA do
        for all b in nodesB do
            path ← AStar(a, b)
            for all node in path do
                if dictionary.ContainsKey(node) then
                    dictionary[node] ← dictionary[node] + 1
                else
                    dictionary[node] = 1
                end if
            end for
        end for
    end for
    return dictionary
end function
```

Algorithm 6 Algorithm to accumulate the resources along a path

```
function ResourceCounter(path)
    resources ← {}
    for all node in path do
        if node.tags[0] ∧ node ∉ resources then
            resources ← resources ∪ node
        end if
        for all route in node.routes do
            if route.a.tags[0] ∧ route.a ∉ resources then
                resources ← resources ∪ route.a
            end if
            if route.b.tags[0] ∧ route.b ∉ resources then
                resources ← resources ∪ route.b
            end if
        end for
    end for
    return resources
end function
```

6 Application

6.1 Setting Up the Model

With the formalization and principles in place, it is possible to examine a first gamespace. As previously discussed in section 2.4, the Summoner's Rift is an ideal example to test the model on. While obtaining a proper 3d-model of the rift is harder than expected, it is not impossible: although Riot Games doesn't provide the model itself, they provide the mini-map, which properly shows the rift. An approximate model of the rift can be obtained by extruding the walls from the mini-map into 3D (see figure 6.2). While this is not perfectly accurate, it is suitable for a first analysis.

The placement of the different nodes went through multiple stages, with different outcomes (e.g. figure 6.1). The goal was to have as little redundant nodes as possible[1] while still maintaining a high accuracy for the distance calculations.

Finally, the model is set up in Unity. All code is written in C#. The centers of nodes are represented by sphere-GameObjects (see figure 6.3). Each node received a script for the node properties. A separate GameObject, the handler, is set up to take care of the main graph functionality: the handler receives a list of connections[2], looks up the according nodes, calculates the routes and node-borders, and stores the distances in the according nodes.

The final model of the rift has a lot more nodes than the early attempts, to increase the precision. To further support the analysis of the rift in the specific context of LoL, off-mesh-links through many of the walls were added: Many champions have mobility spells that allow them to move through walls of a certain size, and every player has the option to equip the summoner spell[3] "flash"[4]. This makes traversing certain walls of the Rift an important and frequent part of the gameplay of LoL, that should be

[1] In this case a node is considered redundant, if there is another node adjacent to it, which has the same properties - the same other nodes are reachable from it, they have identical tags

[2] The connections are passed as triples of node-names: from-via-to. While an automation of this process is possible, this was not the focus of this work.

[3] Summoner spells are active abilities, independent of the champion's abilities. Every player can pick two summoner spells from a pool of different spells with effects like healing, weakening enemies, teleporting or many more

[4] flash allows the player to instantaneously teleport a short distance. Despite its rather long cooldown, flash is the most popular summoner spell in the game

Figure 6.1: An early attempt to distinguish locations for nodes in the jungle

accommodated in the analysis.
The nodes also received tags to further the understanding of the gamespace. The possible tags are: "within the influence of a turret", "contains a jungle camp", "contains a buff camp" and "contains a major buff" (i.e. drake or baron). In a further, in depth, analysis of the game a tag for brushes[5] could be useful.

6.2 Analysing the Rift

With the formalization in place, we can check whether the rift fulfills the principles defined in 4.
For the principles surrounding resource access, the individual values of the resources

[5]Brushes are spaces in the rift with tall grass. Brushes hide units on the inside from enemies on the outside.

Figure 6.2: The base for the 3D-extrusion: mini-map with added towers

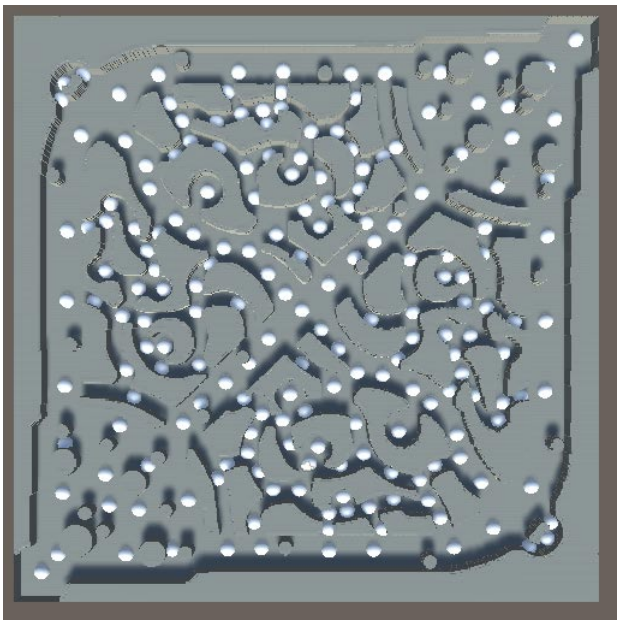


Figure 6.3: The model of the Summoner's Rift with spheres as node centers

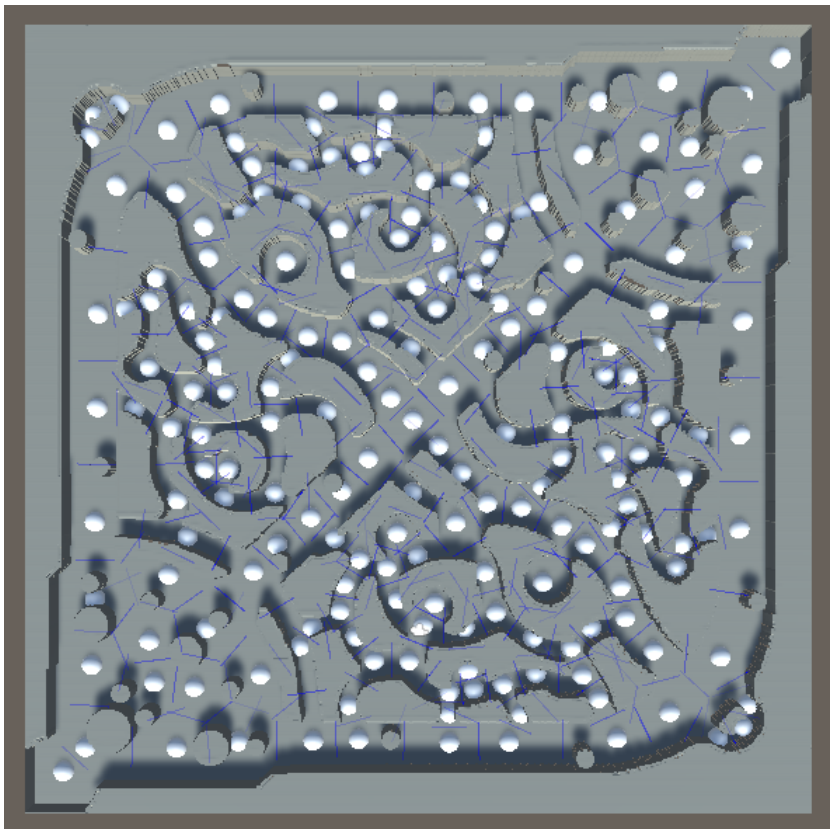


Figure 6.4: The model of the Summoner's Rift, with borders between adjacent nodes

are important. The normal camps are worth less than buff camps, buff camps are worth less than the herald/baron, and the herald/baron is worth less than the drake. These weightings usually change within a game and over the course of different patches, but have proven overall rather consistent for the past two years[6]. Due to the otherwise almost exact symmetry of the rift, the difference in value of the dragon and the baron is quite decisive: Had both of these big objectives the same value, both teams would have equal access and priority on resources.
Simply through the imbalance created by the difference in value of the dragon and the baron, the rift puts the red team at an advantage. The red team has priority on resources of a higher value, a better access score and the possibility to claim resources of higher value in less time than the blue team.
This still doesn't mean that it is impossible for the blue team to claim the dragon (or

[6]Since the big rework of the drakes in 2019 [8], they have consistently been the most important objective in the game, establishing this order of relevance.

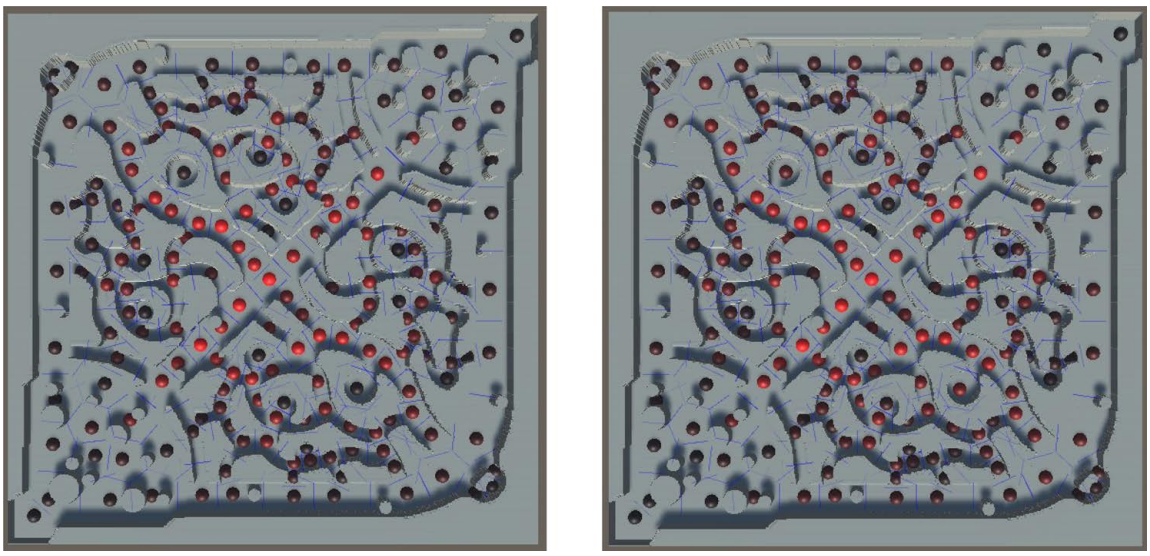


Figure 6.5: finding crossroads: unweighted on the left, weighted with estimated node importance on the right (red nodes are used more often)

the red team to claim the herald/baron); it just makes it harder for them to safely enter the area around it, and to safely fight it. Even if the red team already started fighting it, a player of the blue team may enter the area surprisingly through the wall, by using a mobility spell. The structure of the rift presents merely an advantage in entering the area, not a guarantee to claim the objective.

The central symmetry of the Rift assures a few other properties: the neutral space lies in the middle of the gamespace, equally far away from both bases. Also, as the algorithm for finding crossroads (algorithm 5) confirms (visualized in figure 6.5), the most important crossroad in the rift is exactly in the middle, in the neutral space of the midlane.

Another side effect of this symmetry lies in the mirrored layout of bot- and toplane. The additional gank-path through the tri-bush (labeled as B-Tri and B-2 in figure 6.1) puts the team to whose base it is closer (the red team on the toplane, the blue team on the botlane) at a disadvantage: If a player of said team fights in the middle of the lane, enemies have an additional way to get behind them. This, again, is a case in which the symmetry of the map does not necessarily produce balance. The importance of the drake has led to an increased focus on the botlane, since having a stronger botlaner than the opponent makes it easy to claim drakes (due to the close proximity).

This means that players of almost all roles focus strongly on helping their botlaners[7]. Therefore, as the botlane is generally often more important than the toplane, in this case the symmetry of the gamespace reinforces the advantage of the red team.
It is fairly impossible to lock a team into its base - each team's base has exits through all three lanes, two additional exits only passable for the team that owns the base, and the walls that separate the base from the rest of the rift can be passed through by using a mobility spell (e.g. flash). Even if a team gets pushed deep into their base, under the fountain, it is very hard to spawnkill players (although it becomes possible if one team gained an extreme advantage over the course of the game).
To analyze the range in which resources can be claimed needs to include the range of playable champions[8]. While the radii of the jungle-camps don't overlap (for the individual ranges of different camps refer to the pages on [5]), certain champions can still attack and claim multiple camps at once, due to the champion's own range or mobility. This can make this kind of champion very oppressive as junglers, which is an issue that is usually approached by balancing the champion instead of changing the layout of the jungle or the radius from which resources are claimable.
Finally, the maximal distance between the neutral ground of the lanes is slightly longer than half of the distance from the base to the neutral ground of the bot/toplane. This allows for relatively quick ganks, while preventing an omnipresence of mobile junglers.

6.3 Evaluation

In conclusion it can be said, that the symmetric layout of the Summoner's Rift doesn't make the game perfectly fair. While it prevents multiple vectors of imbalance, the different value of resources at symmetric positions tilts the balance in favor of the red team.
Still, League of Legends has been successful for years now - how is this possible? This is the case, because the gamespace does not entirely determine the fairness of the game. Having a fair gamespace can be a good base for a fair game, but a slightly imbalanced gamespace can be balanced by other components of the game. Especially the draft pick system[9] can significantly benefit one of the teams depending on the current balance of

[7] This has led to strong changes in the meta-game, which not everyone appreciates. Especially toplane players often feel that they lack impact on the game, e.g. `https://www.reddit.com/r/leagueoflegends/comments/k9614z/jungle_is_fundamentally_breaking_this_game_and/` (accessed 2022-01-29)

[8] While other games have resources that can be claimed simply by entering an area, on the Summoner's Rift all resources require the players to attack and kill NPCs to claim them.

[9] The way players select their champions before the game, which involves each player banning a champion, and then taking turns picking champions: blue team picks one - red team picks two - blue team picks

champions[10].
As the current statistics show[11], the current meta-game generally benefits the red team in more organized play (ranked flex, at a rank of platinum+). While this might be mainly due to the current champion balance, the statistics show, that the red team slays more dragons on average (which they have an advantage on, as the analysis of the rift has shown), while the blue team slays more heralds. Interestingly, the red team also slays more barons - the baron spawns at a point in the game at which the early advantage of the red team often has already made them dominant enough to negate their disadvantages towards the baron.
In a further, more in-depth, analysis, it would be especially interesting to analyze the effects of vision[12] and to dive deeper into the topic of mobility spells. This would mean to move away from general principles of fairness, and closer towards the MOBA-genre or LoL itself.

two - red team picks two - blue team picks two - red team picks one

[10] During a period with few very powerful champions, the blue team is at an advantage because it can pick first. During periods with certain strong combos, the red team is at an advantage because they can secure two champions at once.

[11] Current statistics always accessible at `https://www.leagueofgraphs.com/rankings/blue-vs-red/euw`, the text refers to the statistics on 2022-01-30, during patch 12.2 of LoL

[12] A concept that is very significant in the MOBA-genre: The majority of the Rift is covered by fog of war, obscuring any area that is not entered by ally units. Gaining vision on crucial areas is a key part of the gameplay.

7 Conclusion, Discussion, and Future Work

In the end, creating a fair game remains a complex undertaking. The idea to break down the game into conceptual parts, and to start by balancing these appears to be a good way of approaching it. The gamespace is an essential part of a game, and as such is quite interesting in its variety. Setting up a formalized model to represent the topology of the gamespace allows developers to analyze very different gamespaces with respect to the same principles, and can be of great assistance to recognize sources of imbalance in gamespaces.
The principles proposed in this work are very general and applicable to many games, but the framework of the model itself is adaptable and can be complemented by further, more specific tests for each application. Resource access and navigation between different parts of a gamespace are important facets of many competitive multiplayer games, and therefore were the main focus of this work. While the setup of the model takes some time during the engineering process, it can prove to be worth the effort.
There also remains one weakness to the adapted A* algorithm, that can be tackled in various ways in future research or concrete applications: The starting node needs to have stored routes. Hypothetically, the starting node could be positioned in such a way that routes start there, but no route traverses it, meaning that the node itself never stores any route. This can be mitigated in different ways, for instance by choosing a different approach to exploring the first set of nodes, e.g. by looking up nodes that have a stored route that leads to the starting node.
As the analysis of the Summoner's Rift has shown, this way of viewing gamespaces works very well to spot even slight imbalances, assuming the developer has sufficient knowledge of the game. Nonetheless, the analysis of the gamespace alone can't fully determine the fairness of the game: The gamespace is one component among many, and imbalances in one of these might be tipped back into balance by others.
Further work is needed to find further principles, as the ones in this work are largely derived from the MOBA genre - while they are largely applicable to games of other genres, there might be concepts that are more significant in other genres.
There also are multiple alternative approaches to the analysis itself, that can be incorporated in future models. Using a machine-learning approach [13] could be useful for an evaluation of fairness, although it would lead to less human-comprehensible principles, and could easily misguide developers, depending on the training set. On

the other hand, using machine-learning can possibly help to combat human bias in the analysis. Machine-learning could certainly be useful for the development of additional gamespaces for an existing game, checking new gamespaces for similarities to existing ones.
Also, pattern-finding algorithms [11] could prove useful to check gamespaces for symmetric properties or similar paths between different locations, which can be used to control the engineering process, in order to not destroy previously established symmetry or to deliberately establish certain similarities.

List of Figures

List of Algorithms

Bibliography

[1] F. Aurenhammer. "Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure." In: *ACM Comput. Surv.* 23.3 (1991).

[2] B. Crecente. *League of Legends is now 10 years old. This is the story of its birth.* https://www.washingtonpost.com/video-games/2019/10/27/league-legends-is-now-years-old-this-is-story-its-birth/. 2019. (Visited on 01/19/2022).

[3] T. Daniels. *Top 10 highest viewed esports events of 2021*. https://esportsinsider.com/2021/12/highest-viewed-esports-events-2021/. 2021. (Visited on 01/19/2022).

[4] Fandom. *DotA 2 Wiki: Creeps*. https://dota2.fandom.com/wiki/Creeps. (Visited on 01/22/2022).

[5] Fandom. *League of Legends Wiki: Monster*. https://leagueoflegends.fandom.com/wiki/Monster. (Visited on 01/29/2022).

[6] Fandom. *Overwatch Wiki: Health pack*. https://overwatch.fandom.com/wiki/Health_pack. (Visited on 01/22/2022).

[7] J. Funk. *MOBA, DOTA, ARTS: A brief introduction to gaming's biggest, most impenetrable genre*. https://www.polygon.com/2013/9/2/4672920/moba-dota-arts-a-brief-introduction-to-gamings-biggest-most. 2013. (Visited on 01/19/2022).

[8] E. Haas. *Unleashing the Elements*. https://nexus.leagueoflegends.com/en-us/2019/12/unleashing-the-elements/. 2019.

[9] H. Iida. *On games and fairness*. Japan Advanced Institute of Science and Technology, 2007.

[10] N. J. Nilsson. *The Quest for Artificial Intelligence*. Cambridge University Press, 2009.

[11] G. Preti, M. Lissandrini, D. Mottin, and Y. Velegrakis. "Mining patterns in graphs with multiple weights." In: *Distributed and Parallel Databases* 39 (2021).

[12] T. Reddad and C. Verbrugge. *Geometric Analysis of Maps in Real-Time Strategy Games: Measuring Map Quality in a Competitive Setting*. Tech. rep. 3. School of Computer Science, McGill University, 2012.

[13] R. A. Rossi, N. K. Ahmed, R. Zhou, and H. Eldardiry. "Interactive Visual Graph Mining and Learning." In: *ACM Trans. Intell. Syst. Technol.* 9.5 (2018).

[14] J. Schell. *The Art of Game Design: A Book of Lenses, Third Edition*. CRC Press LLC, 2019.

[15] *WELCOME TO THE RIFT LEARN THE BASICS*. https://www.leagueoflegends.com/en-us/how-to-play/. (Visited on 01/19/2022).

[16] Wiktionary. *gamespace — Wiktionary, The Free Dictionary*. https://en.wiktionary.org/w/index.php?title=gamespace&oldid=62030049. 2021. (Visited on 01/20/2022).

[17] M. Wu, S. Xiong, and H. Iida. "Fairness mechanism in multiplayer online battle arena games." In: *2016 3rd International Conference on Systems and Informatics (ICSAI)*. 2016, pp. 387–392. DOI: 10.1109/ICSAI.2016.7810986.