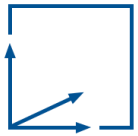
Game Engineering Principles: Fair Gamespaces for Competitive Multiplayer Games

Felix Bartossek

03.03.2022



Final: Bachelor Informatics: Games Engineering

Supervisor: Prof. Gudrun Klinker, Ph.D.

Advisor: Daniel Dyrda, M.Sc

Introduction

- fairness is a core value of playing
- balance to achieve fairness
- more complex game -> harder to balance

→ analyze parts of the game

Introduction

- fairness is a core value of playing
- balance to achieve fairness
- more complex game -> harder to balance

→ analyze parts of the game

- gamespace: the “virtual space” of a video game
- often overlooked in analyses
- symmetry does not solve all problems

Goals

- introduction of principles for fair gamespaces
- creation of a formalization that allows to check these principles
- application of this process to an existing gamespace

Related Work

- research on map quality:
  - T. Reddad and C. Verbrugge, “Geometric Analysis of Maps in Real-Time Strategy Games: Measuring Map Quality in a Competitive Setting”, 2012
- research on fairness in MOBAs:
  - M. Wu, S. Xiong, and H. Iida. “Fairness mechanism in multiplayer online battle arena games.”, 2016

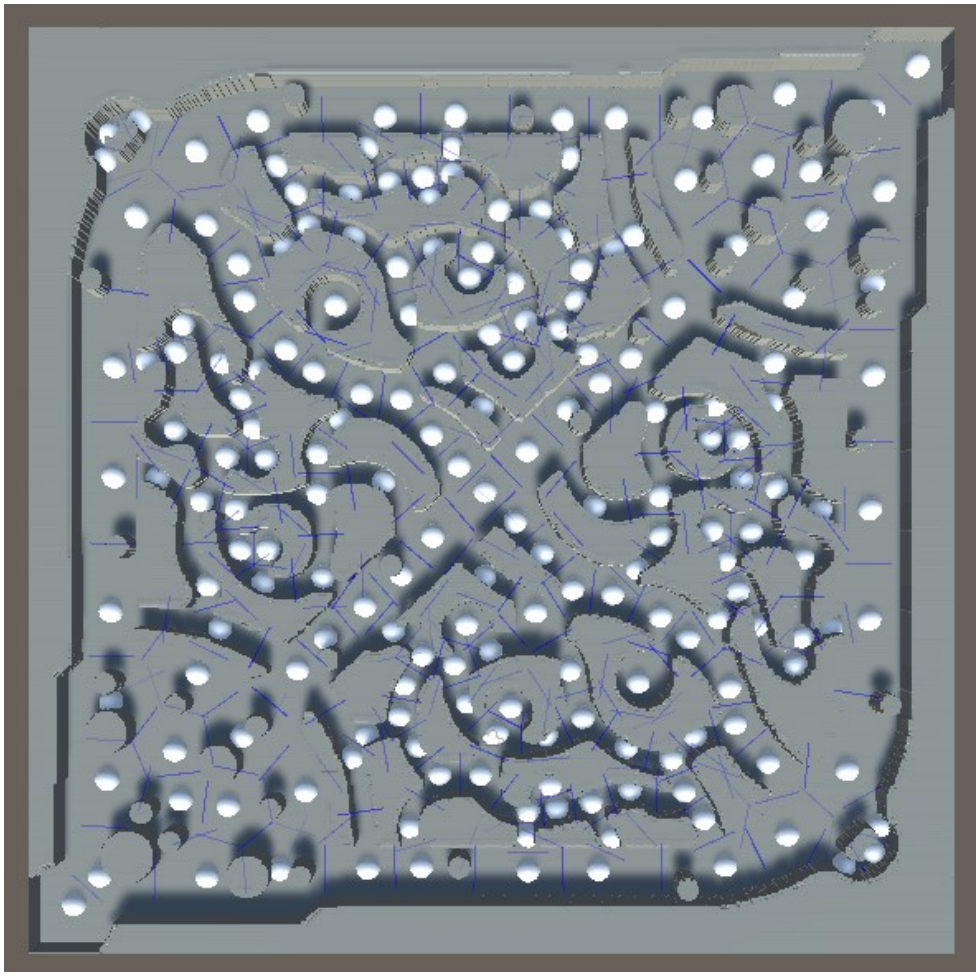
Approach

- formalization necessary to apply analysis to different gamespaces
- important data: topology, resources, importance of areas
- parallel to the engineering process, with design knowledge

Formalization

- representation of gamespace as graph
  - preserves connectivity and distances
- nodes separated similarly to Voronoi partition
  - nodes represent gameplaywise distinct areas
  - every accessible point in the gamespace belongs to a node

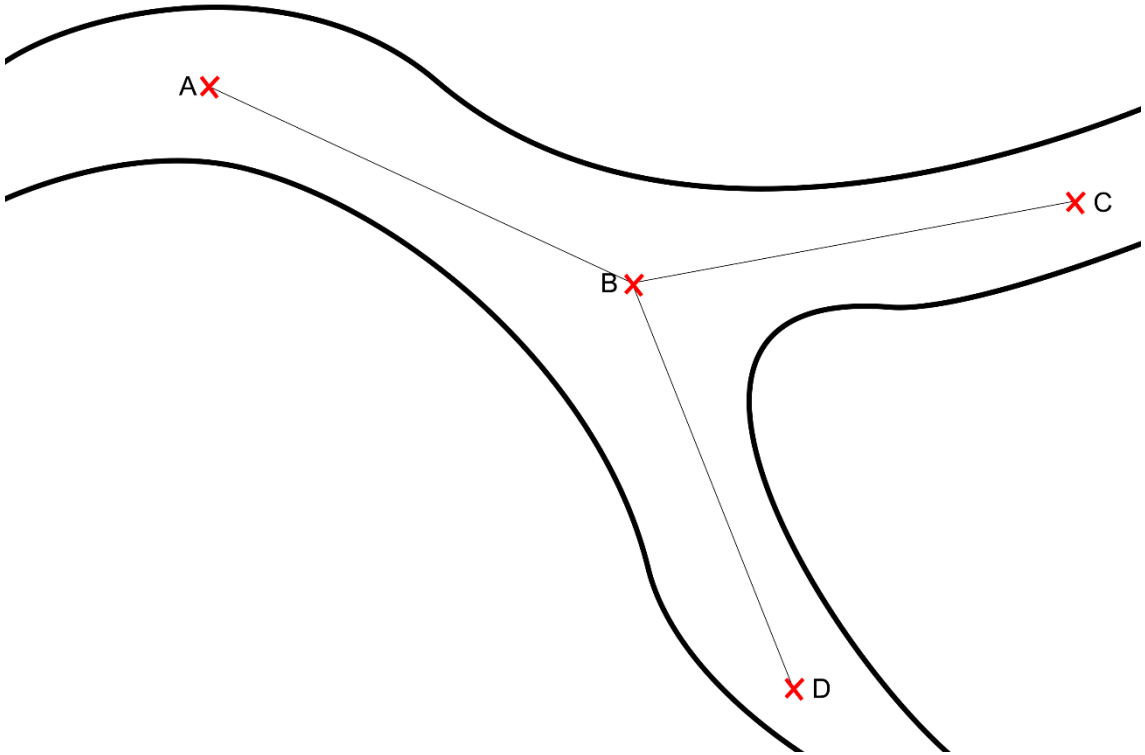
Formalization



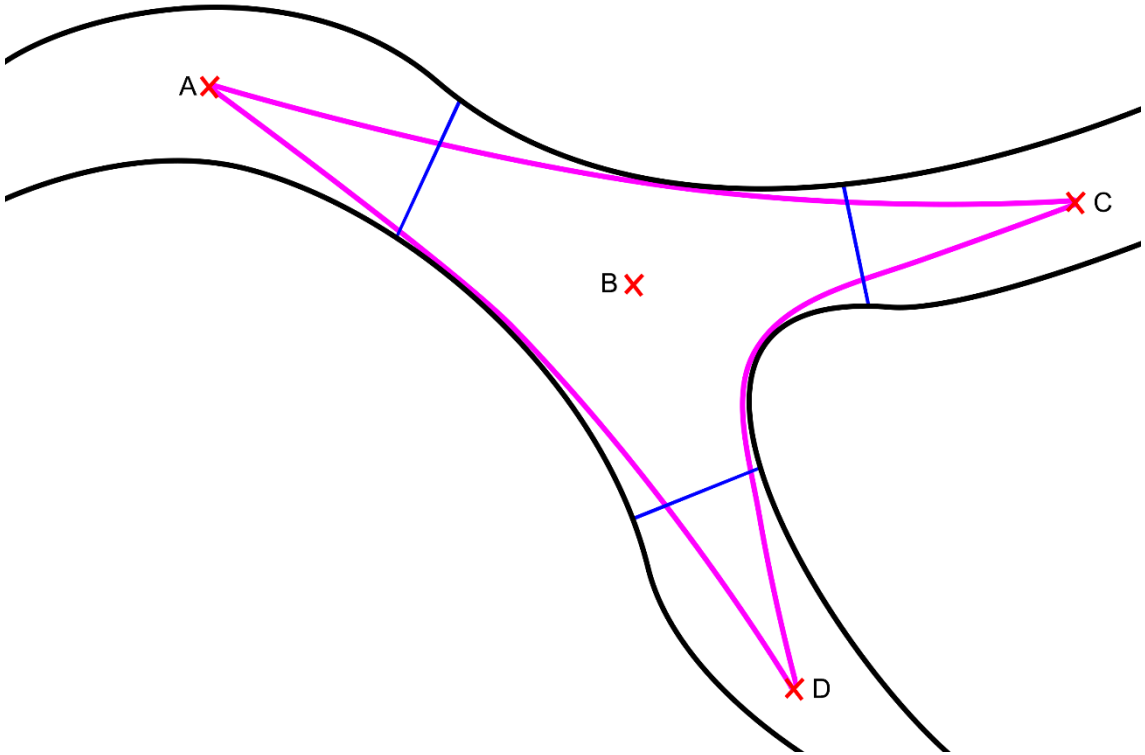
Formalization

- connections between nodes are stored as A via B to C instead of “A to B, B to C”
- a longer path is assembled by only taking contributions of the “via” nodes
  - paths present a lower limit to the actual distance

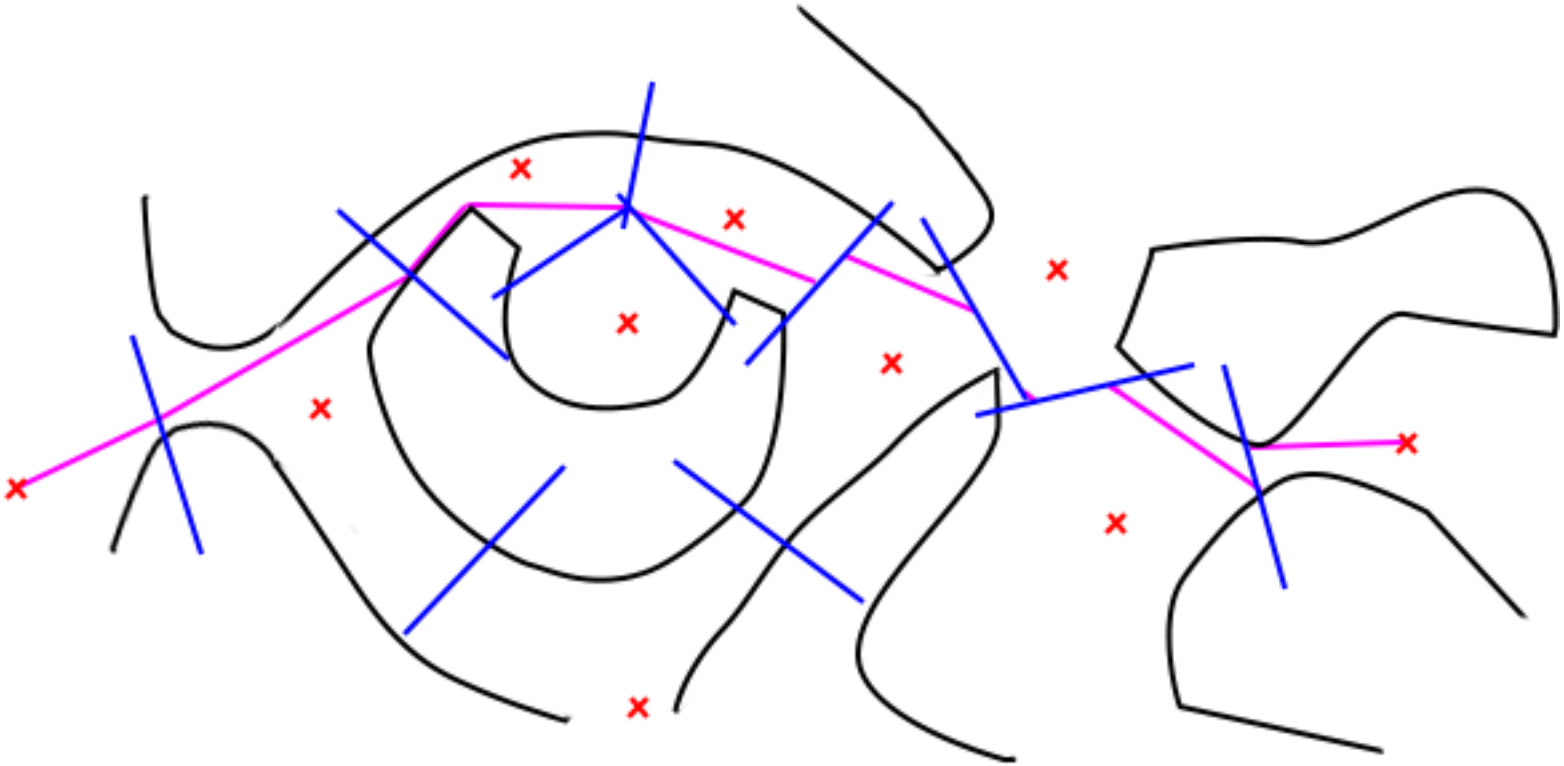
Formalization



Formalization



Formalization



League of Legends

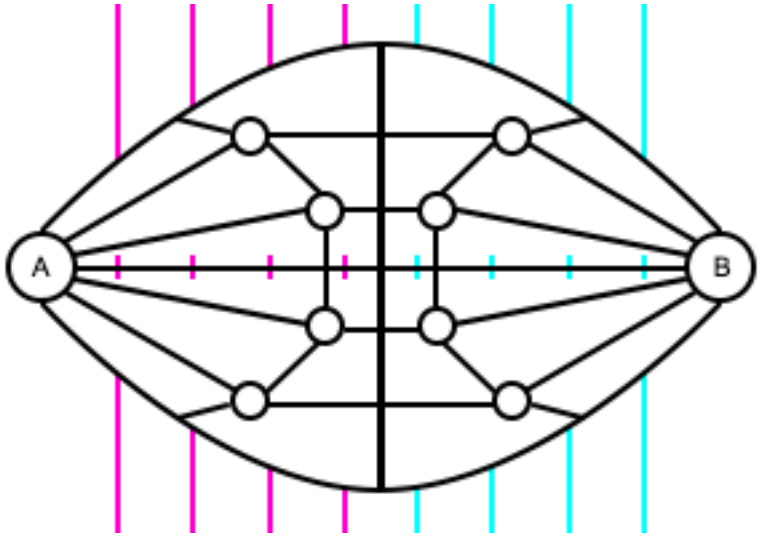
- MOBA by Riot Games, 2009
- continuously updated and extended
- ten players in two teams compete to destroy the enemy base
- highly strategic, both individually and team-wise

League of Legends

Principles

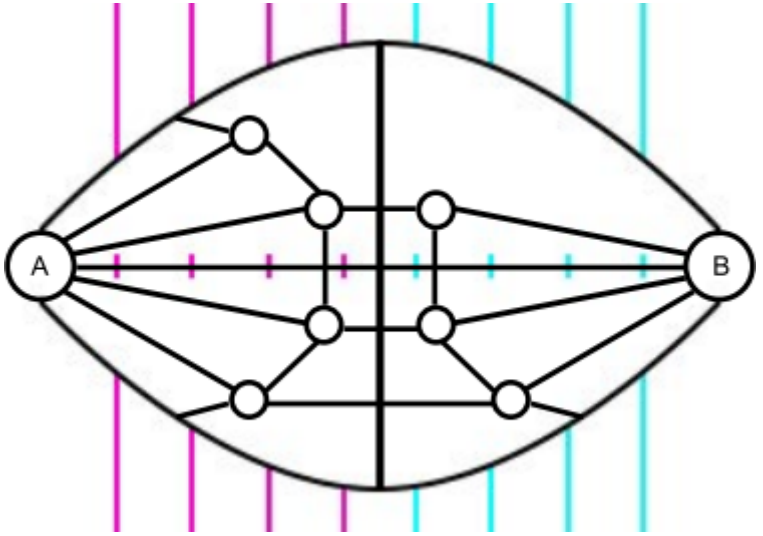
- broadly applicable, although not every principle for every game
- split into two categories:
  - horizontal balance
  - vertical balance

Horizontal Balance



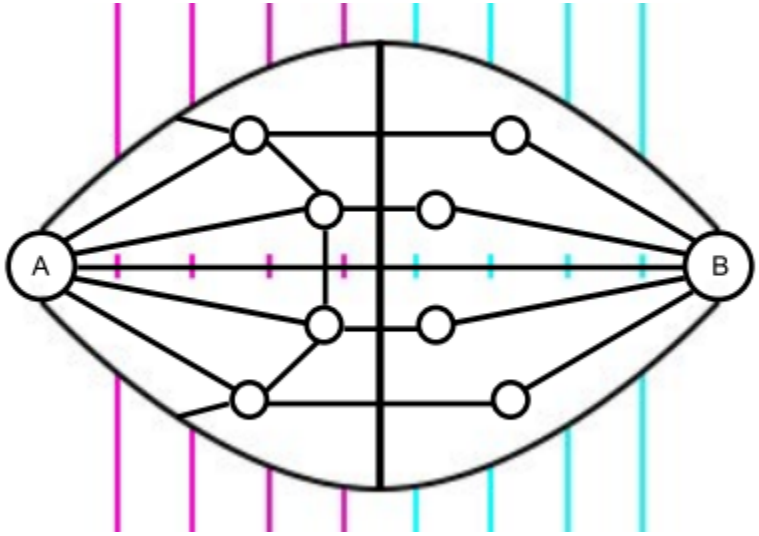
Horizontal Balance

- Every team should have equal opportunities to access resources.



Horizontal Balance

- Every team should have equal opportunities to claim multiple resources one after another

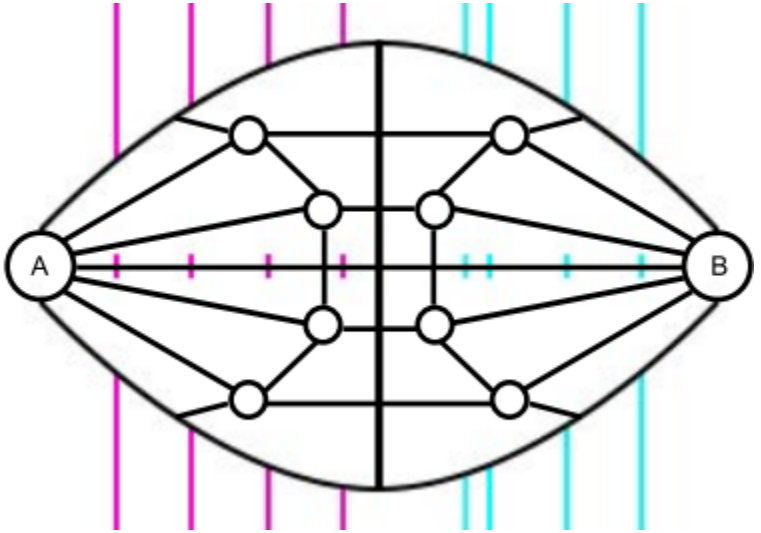


Horizontal Balance

- Every team should have priority on resources of equal value

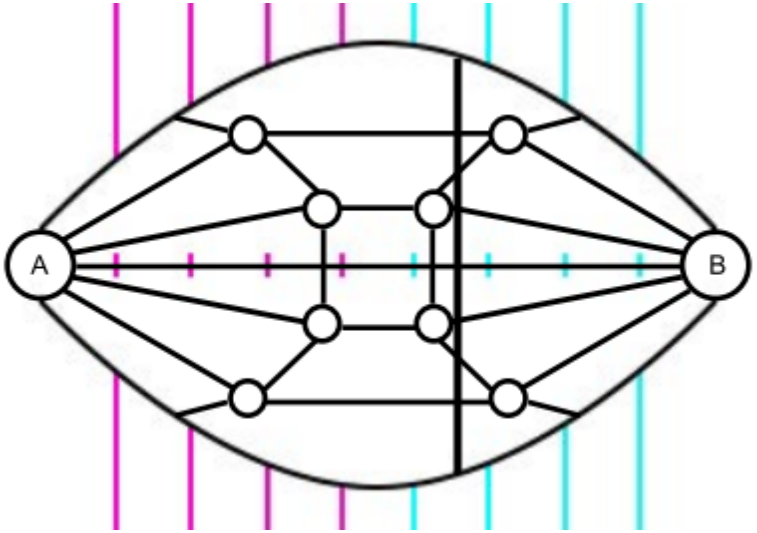
Horizontal Balance

- Neutral territory should, in the beginning of the game, be equally far away from all teams' bases.



Horizontal Balance

- The most common crossroads should lie in neutral territory.



Horizontal Balance

- No base should be locked behind bottlenecks.
- Spawntrapping should not be possible.

Horizontal Balance

- The length and availability of gank-paths should be balanced for both teams.
- The access to objectives along common gank-paths should be balanced.

Vertical Balance

- Players should not be able to claim two resources at once
- Gank-paths should have a set minimal length in relation to the path from the bases to the neutral space

Algorithms

- setting up the routes
  - routes through each point stored within it
  - path is calculated using the engine
  - path is trimmed at the node borders

Algorithms

- setting up the routes
  - routes through each point stored within it
  - path is calculated using the engine
  - path is trimmed at the node borders

- finding paths
  - adapted version of A* algorithm
  - accommodating starting node (no route “through” it)
  - using beeline distance as metric
  - distances always with respective predecessor
  - returns path
  - separate lookup function to get length

Algorithms

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
```

Algorithms

- measuring access:

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

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
```

Algorithms

- measuring priority:

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
```

Algorithms

- finding crossroads:

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
```

Algorithms

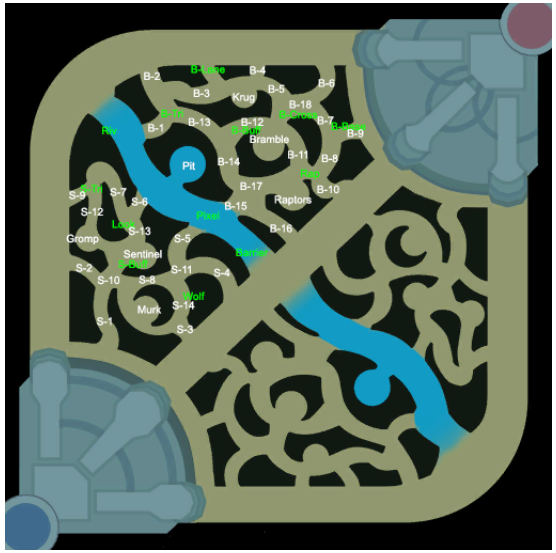
- finding resources along a gank-path:

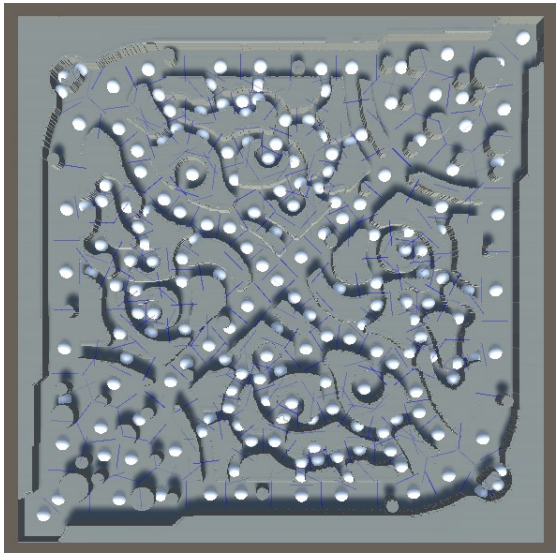
Algorithm 6 Algorithm to accumulate the resources along a path

```
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
```

Application

- setup
  - no official model of the Summoner's Rift available
  - instead own 3D-extrusion from the mini map





Application

- setup
  - no official model of the Summoner's Rift available
  - instead own 3D-extrusion from the mini map
  - careful placement of nodes, using existing knowledge
  - setup in unity, all code written in C#
  - model accommodates traversing through walls

Application

- analysis
  - central symmetry of the Rift ensures many properties
  - → assumption: if resources at equal positions are equally important, it is fair
  - not the case!
  - → more priority for red team
  - → better access score for red team
  - → better gank-paths for the red team

  - almost no spawntrapping
  - claiming two resources at a time possible
  - appropriate length of gank-paths

Application

- evaluation
  - Summoner's Rift favors red team
  - still: can be balanced via other factors
  - current statistics align with analysis:
    - in organized play: slightly higher win-rate for red team
    - blue team kills more heralds
    - red team kills more dragons
    - red team kills more barons due to early advantages

Conclusion

- formalized gamespace is useful for balancing it
- fair gamespace alone does not create a fair game, and vice versa
- proposed principles are very general and adaptable

Further Work

- more principles (e.g. from other genres)
- eliminate weaknesses in the node system and pathfinding
- further interesting approaches to the topic:
  - machine learning
  - pattern-finding algorithms

References

- F. Aurenhammer. “Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure.” In: ACM Comput. Surv. 23.3 (1991)
- 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- isnow-years-old-this-is-story-its-birth/. 2019. (Visited on 01/19/2022).
- 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-abrief-introduction-to-gamings-biggest-most. 2013. (Visited on 01/19/2022)
- H. Iida. On games and fairness. Japan Advanced Institute of Science and Technology, 2007.
- 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.
- WELCOME TO THE RIFT LEARN THE BASICS. https://www.leagueoflegends.com/en-us/how-to-play/. (Visited on 01/19/2022).
- Wiktionary. gamespace — Wiktionary, The Free Dictionary. https://en.wiktionary. org/w/index.php?title=gamespace&oldid=62030049. 2021. (Visited on 01/20/2022)
- 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.

- Most importantly: J. Schell. The Art of Game Design: A Book of Lenses, Third Edition. CRC Press LLC, 2019.