

Универзитет у Београду
Математички факултет



НИКОЛА СТАНОЈЕВИЋ

Алгоритми за проналажење
најкраћег пута у видео играма

МАСТЕР РАД

Ментор: Миодраг Живковић

Београд,
2017.

Садржај

Глава 1

Увод

Глава 2

Начини представљања

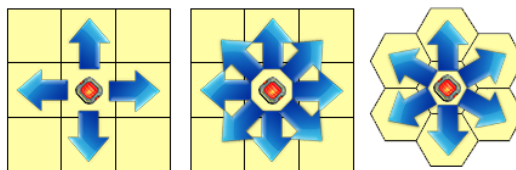
КОМПЛЕКСНИХ СВЕТОВА ВИДЕО

ИГАРА

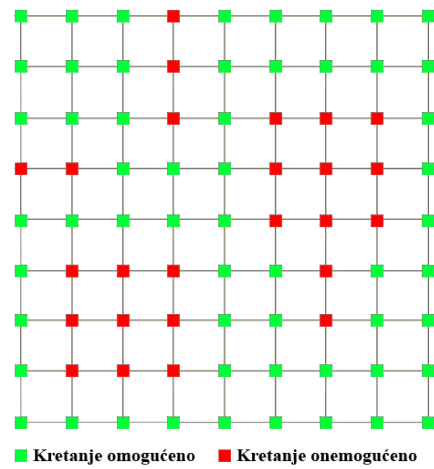
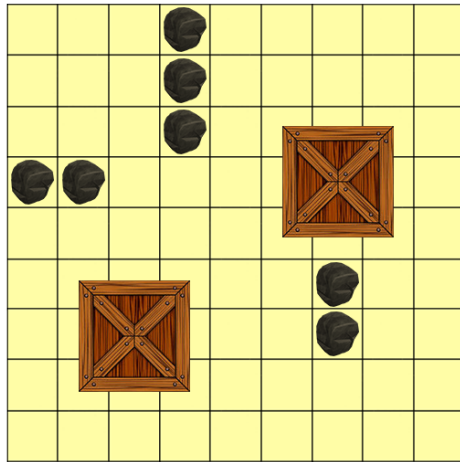
2.2.1 Навигациони графови засновани на мрежи

RTS real time strategy

RTS

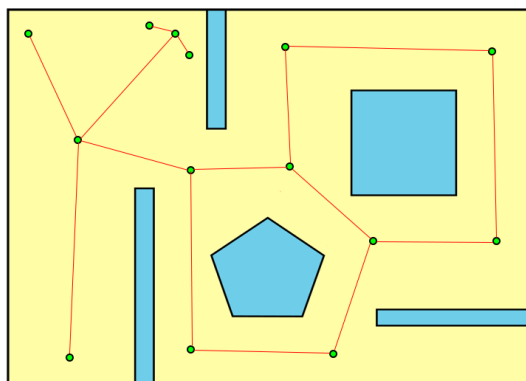


Пло и е ко е омогу у у крета е у егири осам или ест пра-
ва а



Мапа игре лево и њен одговарајући навигациони граф десно

2.2.2 Навигациони графови засновани на маркерима



Пуно креиран навигациони граф заснован на маркерима

points of visibility

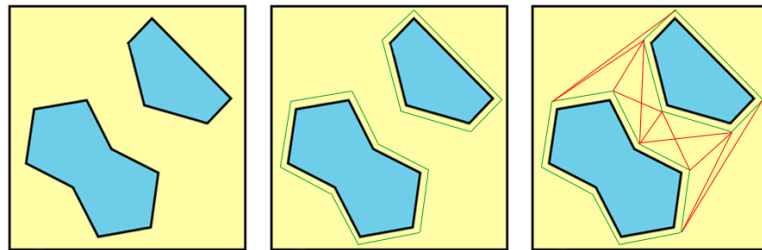
POV
POV

POV

POV

POV

POV

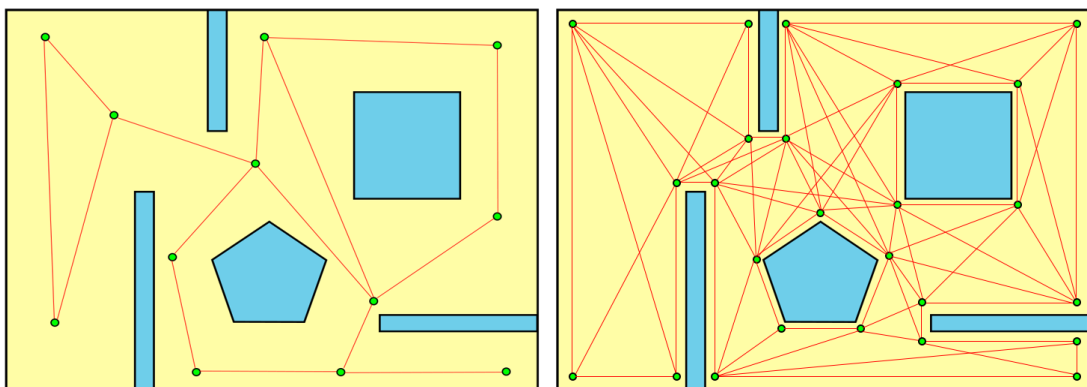


Про ес про ирива а об еката и генериса а навига ионог гра а

POV

POV

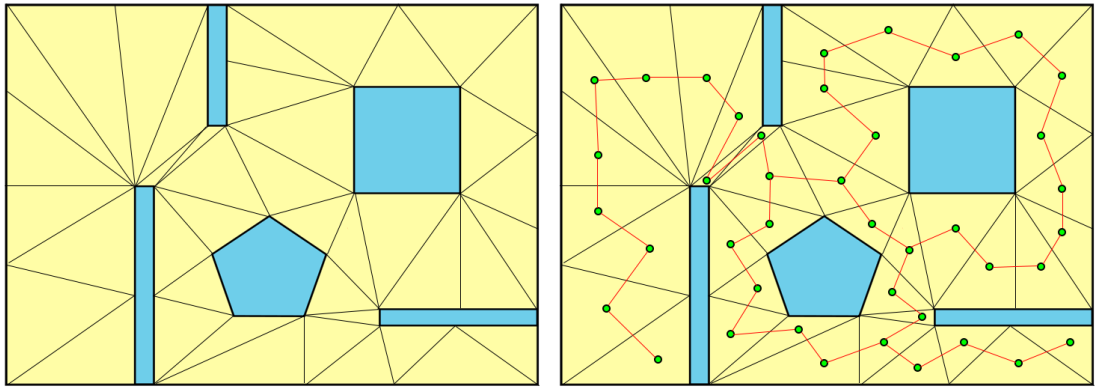
POV



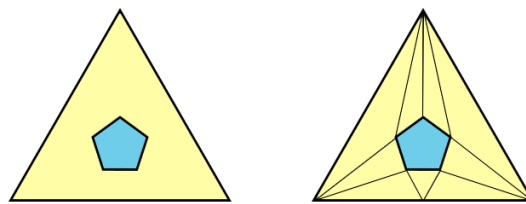
POV гра са ру но одабраним маркерима лево и POV гра креиран кори е ем те ника про ирене геометри е десно

POV

2.2.3 Навигациони графови засновани на мрежама конвексних полигона



Мапа представена помоу конвексни полигона лево и ена одговарау и навига иони гра десно



Промена поделе на полигоне након додава а новог об екта на мапи

Глава 3

Захтеви и ограничења алгоритама за проналажење најкраћег пута у видео играма

ocking

path smooting

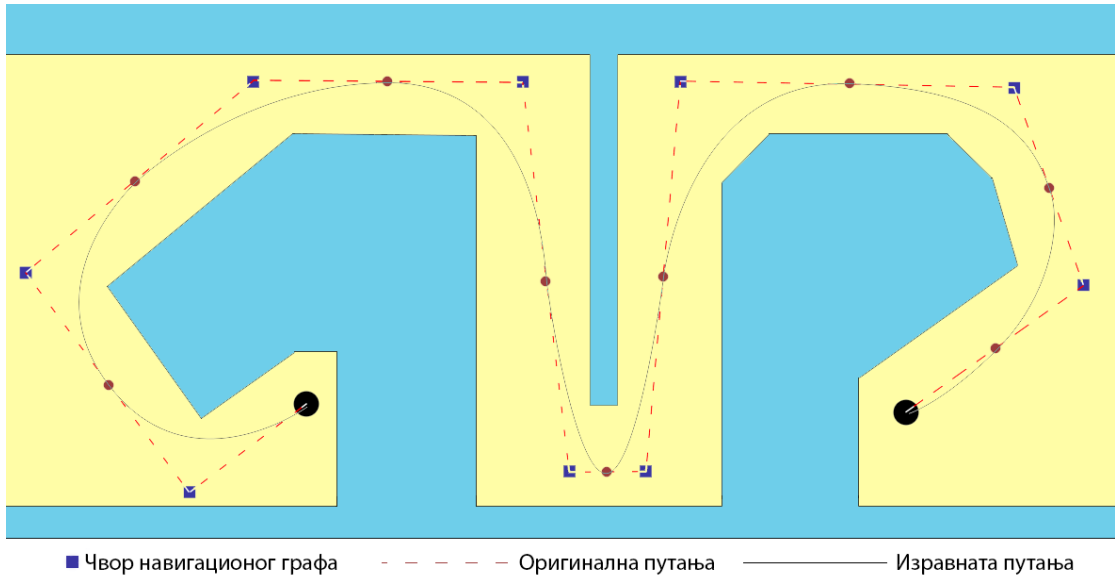
3.2.1 Оптималност путање

$$\text{Procenat optimalnosti} = \frac{S_{predlozeno} \cdot S_{optimalno}}{S_{optimalno}}$$

$S_{predlozeno}$

$S_{optimalno}$

3.2.2 Заобљавање путање



Ефекат заобилавања путање резултује у путање добијене преко навигационог графа.



RTS - real time strategy

RTS

top-down

3.3.1 Временска ограничења перформанси

3.3.2 Меморијска ограничења

25MB

2MB

1MB

- Hierarchical Path-Finding A*

HPA*



Company of Heroes

gameplay

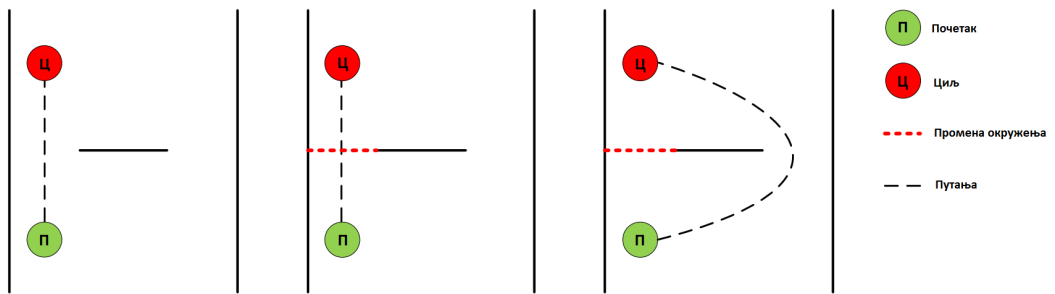
Company of Heroes Worms Battlefield



Пример измене изгледа окруже а игре у видео игри Company of Heroes По етно ста е окруже а игре лево и исто окруже е након битке десно

Battlefield

Battlefield Hardline



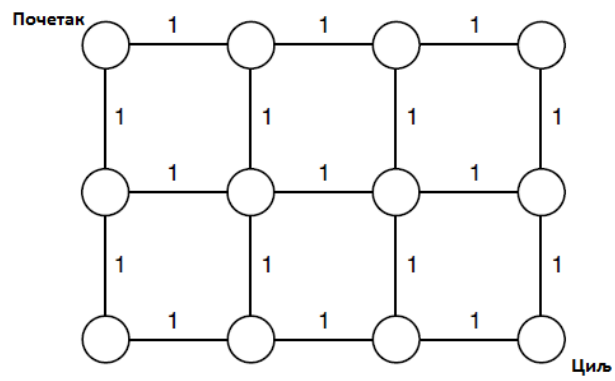
Репланира е пута е услед промене у окруже у По етна
 пута а лево прекид по етне пута е средина и репланирана пута а
 десно

response time

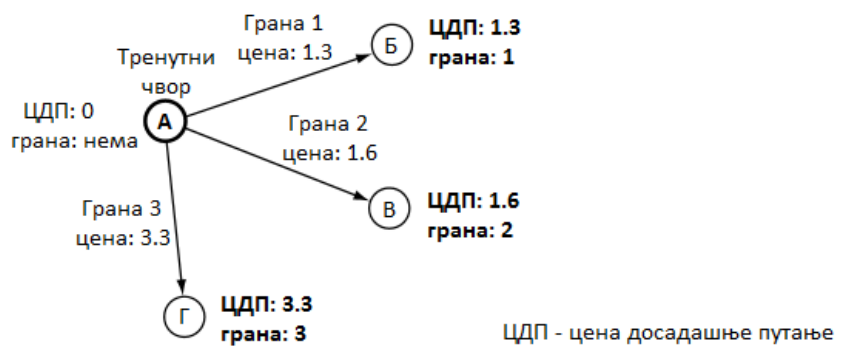
Глава 4

Најчешће коришћени алгоритми
за проналажење најкраћег пута
у видео играма

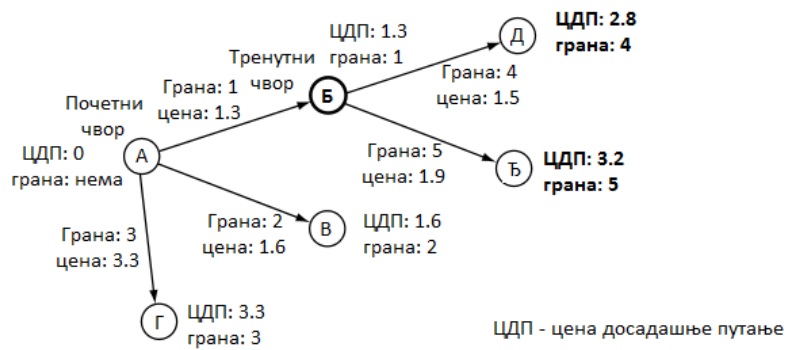
4.1.1 Дијкстрин алгоритам



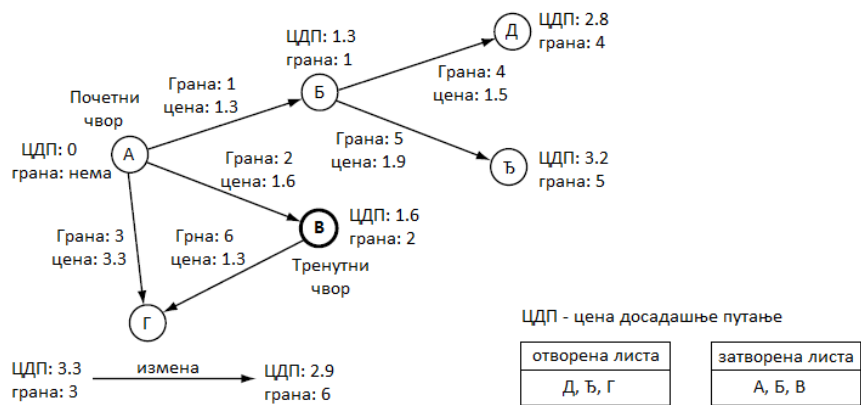
Ви е на кра и пута а са истом еном изме у по етног и и ног вора



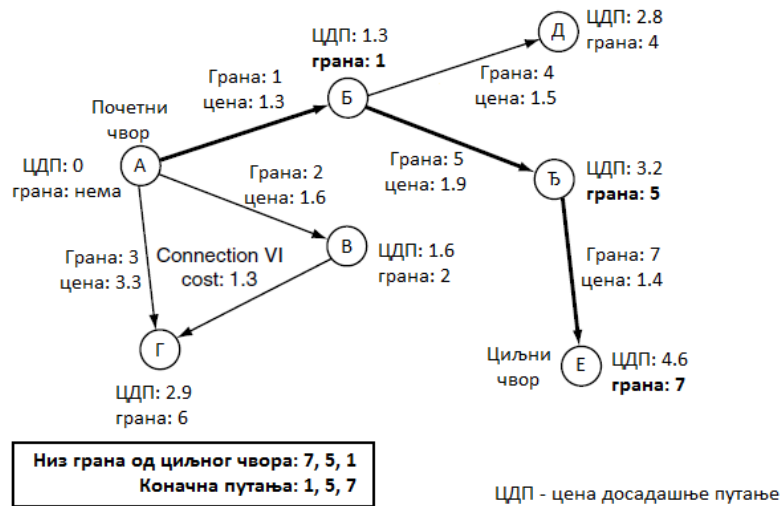
Дијекстрин алгоритам у првој итерацији и где је постојећи чвор уједињен и тренутни чвор



Ди кстрин алгоритам у друго итера и и



Измена ене досада е пута е на отвореном вору



Реконструкција на крајње путање

```

1 def pathfindDijkstra(graph, start, end):
2
3 # Структура која се користи ради чувања
4 # поратака у о сваком од чворова
5 struct NodeRecord:
6 node
7 connection

```

```

8 costSoFar
9
10 # Inicijalizacija podataka za pocetni cvor
11 startRecord = new NodeRecord()
12 startRecord.node = start
13 startRecord.connection = None
14 startRecord.costSoFar = 0
15
16 # Inicijalizacija otvorene i zatvorene liste cvorova
17 open = PathfindingList()
18 open += startRecord
19 closed = PathfindingList()
20
21
22 # Obrada cvorova kroz iteracije
23 while length(open) > 0:
24
25 # Pronaci element sa najnižom vrednoscu iz otvorene liste
26 # (posmatrajuci cenu dosadasnje putanje cvorova)
27 current = open.smallestElement()
28
29 # Ukoliko je trenutni cvor jednak ciljnom cvoru
30 # prekinuti izvr avanje algoritma.
31 if current.node == goal: break
32
33 # U suprotnom pronaci grane ka drugim cvorovima od trenutnog cvora.
34 connections = graph.getConnections(current)
35
36 # Proci kroz sve grane koje su pronadjene.
37 for connection in connections:
38
39 # Pokupiti procenu cene za ciljni cvor
40 endNode = connection.getToNode()
41
42 endNodeCost = current.costSoFar +
43 connection.getCost()
44
45 # Preskociti cvor ako je zatvoren
46 if closed.contains(endNode): continue
47
48 # .. ili ako je otvoren a pronadjena je
49 # losija putanja
50 else if open.contains(endNode):
51

```

```

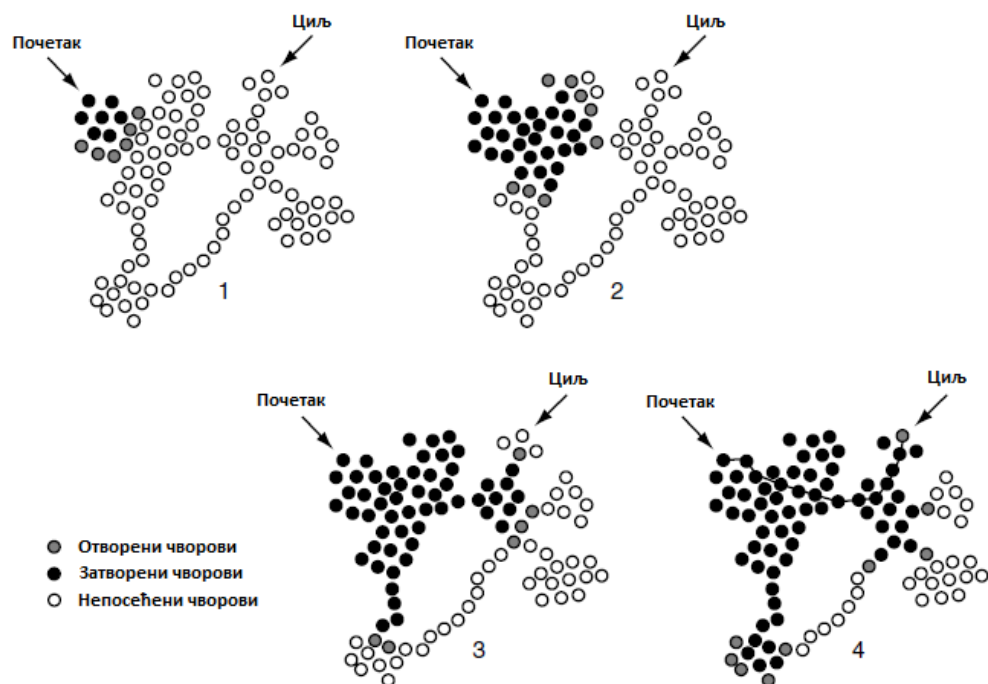
52 # Pronalazenje podataka u otvorenoj listi
53 # koji odgovaraju ciljnom cvoru.
54 endNodeRecord = open.find(endNode)
55
56 if endNodeRecord.cost <= endNodeCost:
57     continue
58
59 # U suprotnom imamo neposecen cvor
60 # pa se za njega cuvaju podaci
61 else:
62     endNodeRecord = new NodeRecord()
63     endNodeRecord.node = endNode
64
65 # Ukoliko je potrebno azurirati podatke
66 # Azurira se cena i grana
67 endNodeRecord.cost = endNodeCost
68 endNodeRecord.connection = connection
69
70 # Dodati cvor u otvorenu listu
71 if not open.contains(endNode):
72     open += endNodeRecord
73
74 # Nakon sto je završen prolayak kroz grane
75 # trenutnog cvora, dodati cvor u zatvorenu listu
76 # i ukoloniti cvor iz zatvorene liste.
77 open = current
78 closed += current
79
80 # Ukoliko trenutni cvor nije ciljni cvor
81 # ili kada nema vise cvorova za pretragu
82 if current.node != goal:
83
84 # Ponestalo je cvorova a resenje
85 #nije pronadjeno.
86 return None
87
88 else:
89 # Napraviti listu u koju je
90 # potrebno smestiti konacnu putanju
91 path = []
92
93 # Vracajuci se unazad formirati konacnu putanju,
94 # sakupljajuci grane u listu
95 while current.node != start:

```

```

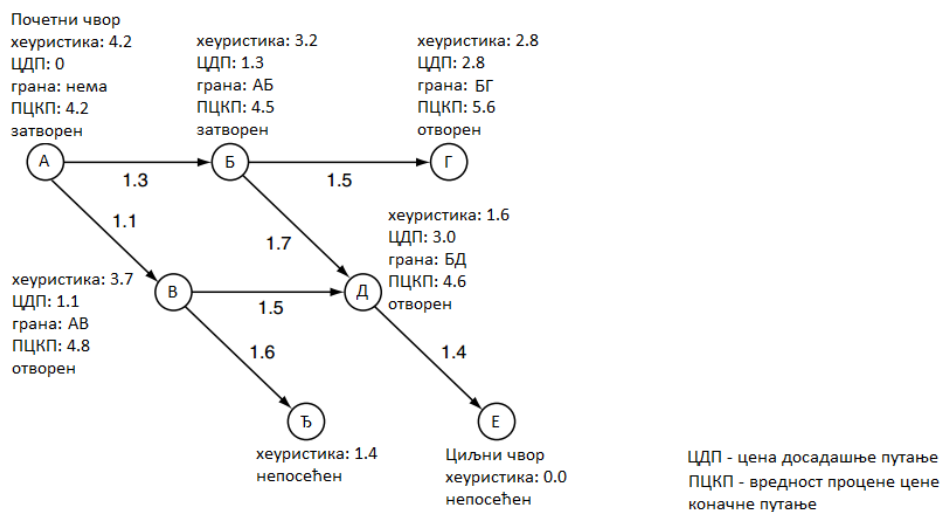
96 path += current.connection
97 current = current.connection.getFromNode()
98
99 # Obrnuti listu kako bi sadzala grane
100 # konacne putanje sortirane od pocetnog
101 # ka ciljnom cvoru.
102 # Vratiti listu kao rezultat.
103 return reverse(path)

```

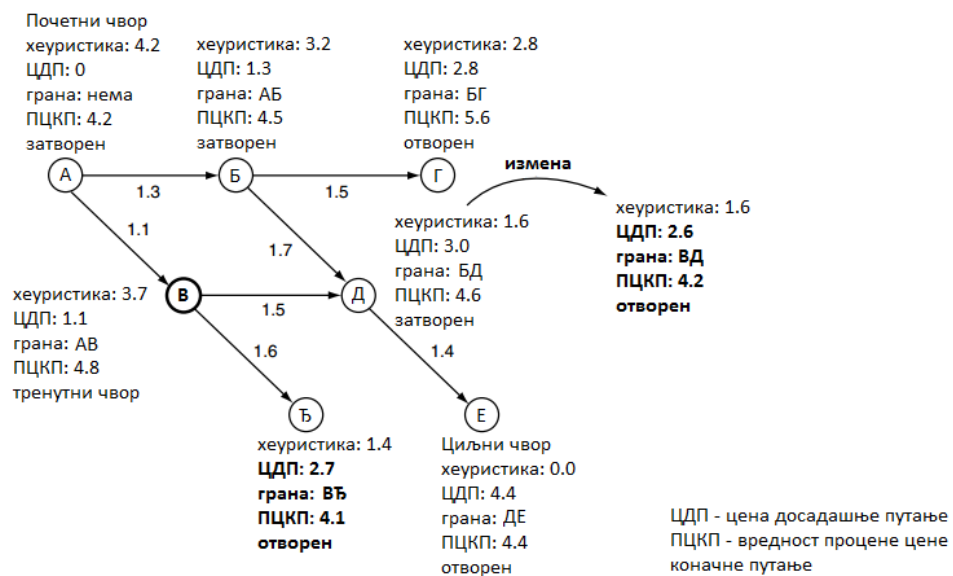


Обилазак чворова у неколико корака

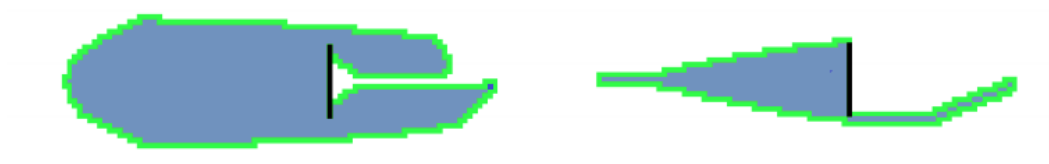
4.1.2 Алгоритм A^*



А са вредностима процене укупне цене коначне путање



Ажурира е затвореног вора



Простор претраге кори је ем унк и е еуристике ко а под-
е у е лево и унк и е еуристике ко а пре е у е десно дужину пута е
до и ног вора

Deepening A*

Iterative

Simplified Memory-Bounded A*

Fringe Search

Lifelong Planning A*

D* Lite

```
1 def pathfindAStar(graph, start, end, heuristic):
2
3 # Struktura koja se koristi radi cuvanja
4 # porataka u o svakom od cvorova
5 struct NodeRecord:
6 node
7 connection
8 costSoFar
9 estimatedTotalCost
10
```

```

11 # Inicijalizacija podataka za pocetni cvor
12 startRecord = new NodeRecord()
13 startRecord.node = start
14 startRecord.connection = None
15 startRecord.costSoFar = 0
16 startRecord.estimatedTotalCost =
17 heuristic.estimate(start)
18
19 # Inicijalizacija otvorene i zatvorene liste cvorova
20 open = PathfindingList()
21 open += startRecord
22 closed = PathfindingList()
23
24 # Obrada cvorova kroz iteracije
25 while length(open) > 0:
26
27 # Pronaci element sa najnižom vrednoscu iz otvorene liste
28 # (posmatrajuci vrednost procene ukupne cene konacne putanje)
29 current = open.smallestElement()
30
31 # Ukoliko je trenutni cvor jednak ciljnom cvoru
32 # prekinuti izvršavanje algoritma.
33 if current.node == goal: break
34
35 # U suprotnom pronaci grane ka drugim cvorovima od trenutnog cvora.
36 connections = graph.getConnections(current)
37
38 # Za svaku od datih grana trenutnog cvora...
39 for connection in connections:
40
41 # ... pokupiti vrednost procene ukupne cene
42 # cvorova povezanih tim granama sa trenutnim cvorom
43 endNode = connection.getToNode()
44 endNodeCost = current.costSoFar +
45 connection.getCost()
46
47 # Ukoliko je cvor zatvoren, potrebno ga je
48 # preskociti ili ukloniti iz zatvorene liste.
49 if closed.contains(endNode):
50
51 # Pronalaze se podaci iz zatvorene liste
52 # koji se odnose na ciljni cvor
53 endNodeRecord = closed.find(endNode)
54

```

```
55 # Ukoliko pronadjena putanja nije kraca , nastaviti dalje
56 if endNodeRecord.costSoFar <= endNodeCost:
57     continue;
58
59 # U suprotnom ukloniti ciljni cvor iz zatvorene liste
60 closed = endNodeRecord
61
62 # Stare vrednosti cvora mogu se iskoristiti
63 # radi proračuna heuristike bez pozivanja
64 # potencijalno skupe funkcije heuristike
65 endNodeHeuristic = endNodeRecord.cost
66 endNodeRecord.costSoFar
67
68 # Preskoci ako je cvor otvoren i
69 # nije pronadjena bolja putanja
70 else if open.contains(endNode):
71
72 # Pronalazenje podataka u otvorenoj listi
73 # koji odgovaraju ciljnom cvoru
74 endNodeRecord = open.find(endNode)
75
76 # Ukoliko putanja nije bolja , preskoci
77 if endNodeRecord.costSoFar <= endNodeCost:
78     continue;
79
80 # Stare vrednosti cvora mogu se iskoristiti
81 # radi proračuna heuristike bez pozivanja
82 # potencijalno skupe funkcije heuristike
83 endNodeHeuristic = endNodeRecord.cost
84 endNodeRecord.costSoFar
85
86 # U suprotnom imamo neposecen cvor
87 # potrebno je popuniti podatke za njega
88 else:
89     endNodeRecord = new NodeRecord()
90     endNodeRecord.node = endNode
91
92 # Potrebno je izracunati vrednost heuristike
93 # koristeći funkciju , posto ne postoje stari
94 # podatci o cvoru koji bi mogli biti iskorisceni
95 endNodeHeuristic = heuristic.estimate(endNode)
96
97 # Ukoliko je potrebno azurirati podatke o cvoru
98 # Azurirati cenu , vrednost procene i granu
```

```
99 endNodeRecord.cost = endNodeCost
100 endNodeRecord.connection = connection
101 endNodeRecord.estimatedTotalCost =
102 endNodeCost + endNodeHeuristic
103
104 # I dodati cvor sa podacima u otvorenu listu
105 if not open.contains(endNode):
106 open += endNodeRecord
107
108 # Završen prolazak kroz grane trenutnog cvora
109 # Dodati cvor u zatvorenu listu
110 # i ukloniti ga iz otvorene liste
111 open = current
112 closed += current
113
114 # Ukoliko trenutni cvor nije ciljani cvor
115 # ili kada nema više cvorova za pretragu
116 if current.node != goal:
117
118 # Ponestalo je cvorova, a rešenje nije pronađeno
119 return None
120
121 else:
122
123 # Napraviti listu u koju je
124 # potrebno smestiti konačnu putanju
125 path = []
126
127 # Vracajući se unazad formirati konačnu putanju,
128 # sakupljajući grane u listu
129 while current.node != start:
130 path += current.connection
131 current = current.connection.getFromNode()
132
133 # Obrnuti listu kako bi sadržala grane
134 # konačne putanje sortirane od početnog
135 # ka ciljnom cvoru.
136 # Vratiti listu kao rezultat.
137 return reverse(path)
```

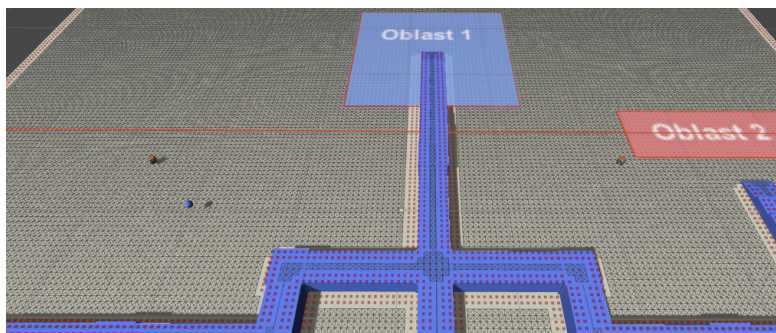
Глава 5

Примери и тестови

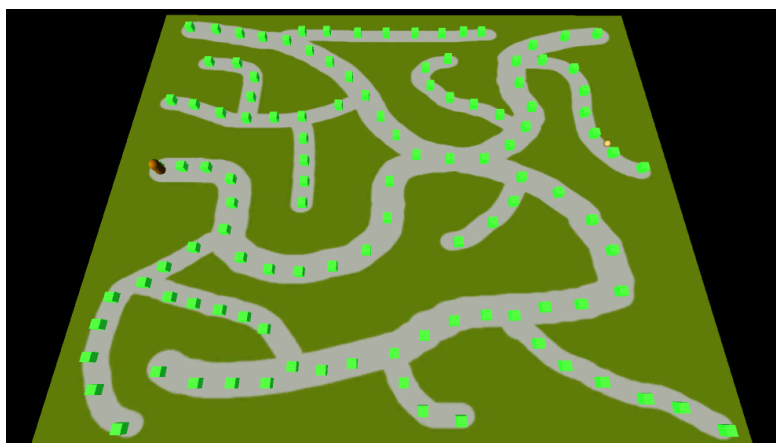
Unity

game

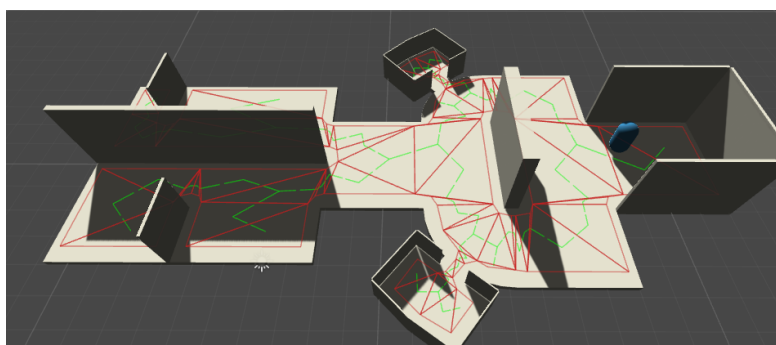
engine OS X



Мапа са навига ионим гра ом заснованом на мрежи



Мапа са навига ионим гра ом заснованом на маркерима

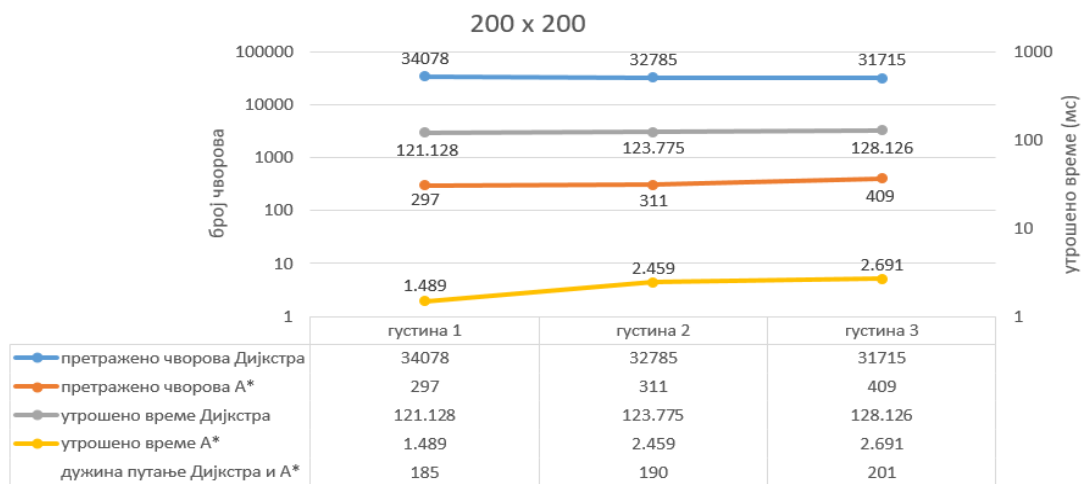


Мапа са навига ионим гра ом заснованим на мрежама кон-
вексни полигона

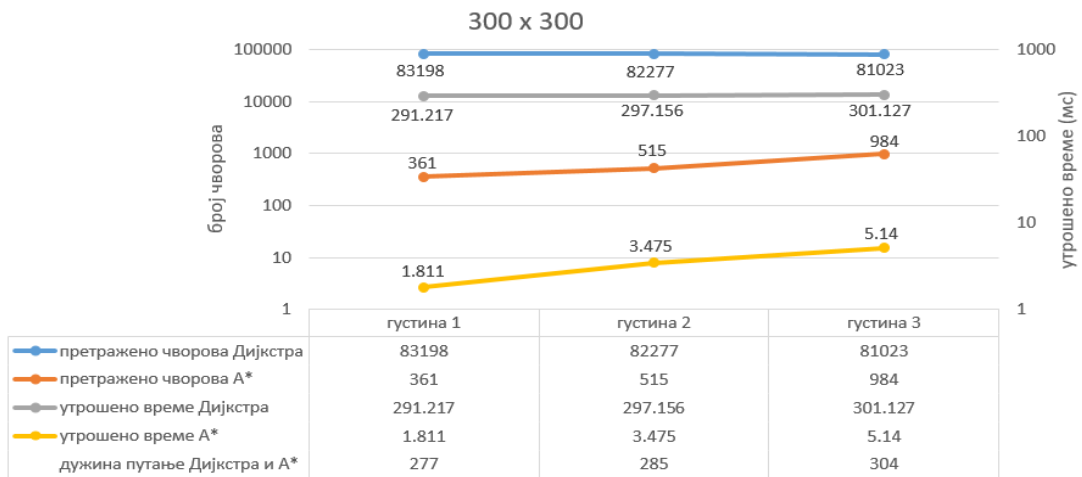
5.2.1 Опис теста

200 300 500

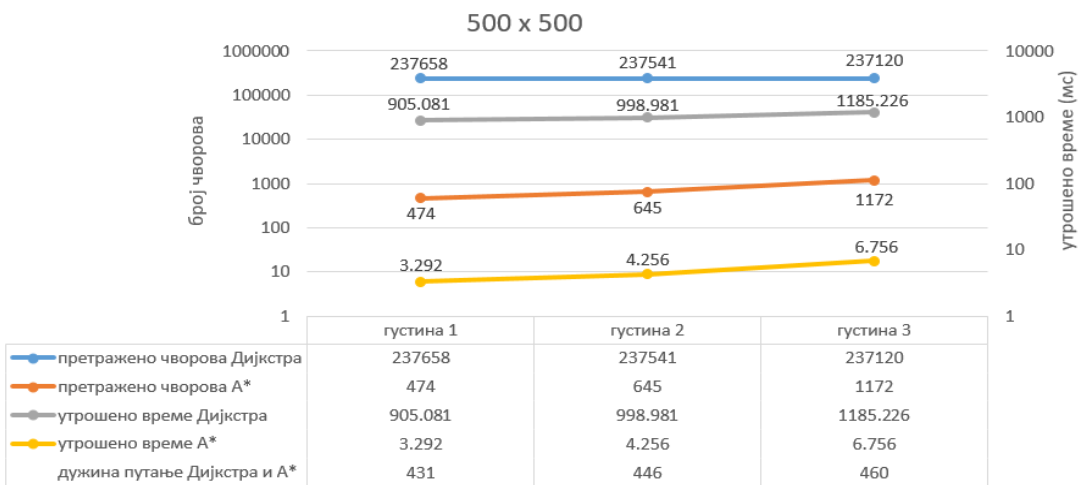
5.2.2 Резултати теста



Доби ени резултати за димензи у мапе

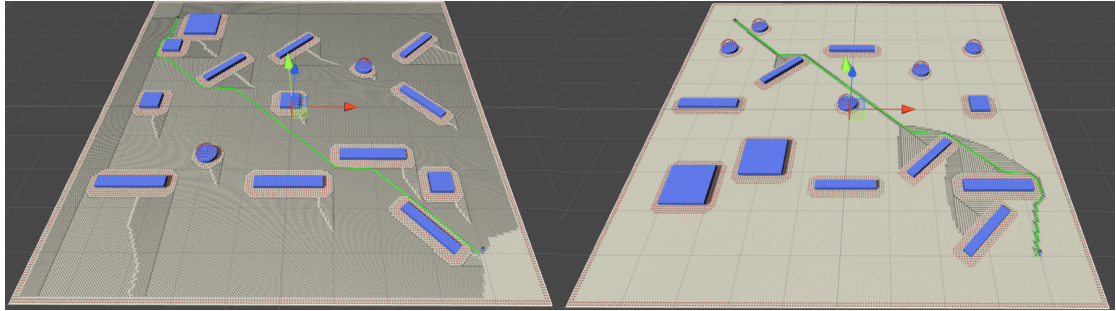


Доби ени резултати за димензи у мапе



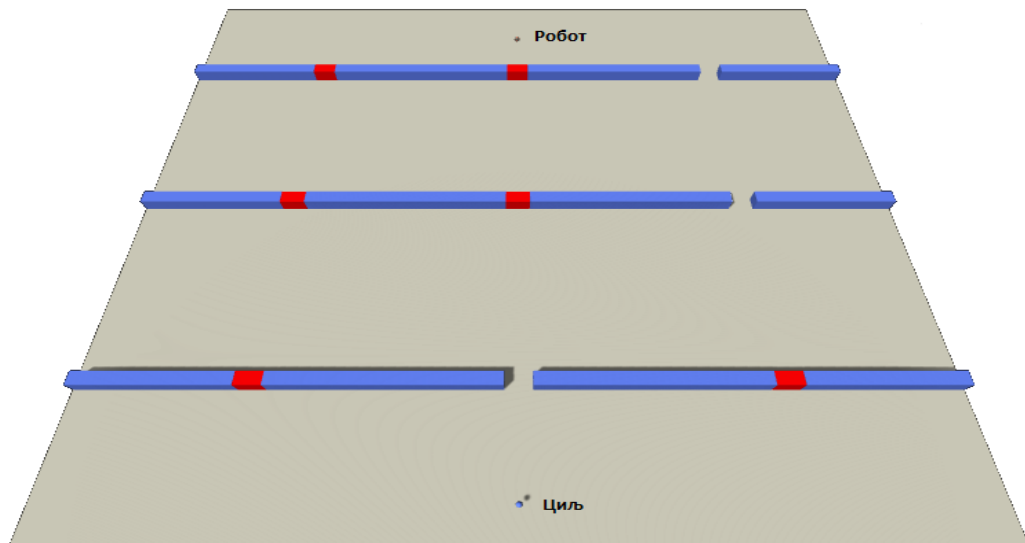
Доби ени резултати за димензи у мапе

5.2.3 Анализа резултата теста



Поре е е испу ености простора претраге кори е ем Ди к-стриног алгоритма лево и кори е ем алгоритма А десно

5.3.1 Опис примера



Пример динами ке промене мале са зидовима и вратима

Глава 6

Закључак

Прилог: речник термина

Кретање у јату (енг. osking)

Граф

Полигон

Вештачка интелигенција (енг. arti cial intelligence)

Механика извођења игре (енг. gameplay)

Атомична акција

Фрејм

FPS - frames per second

Време одзива (енг. response time)

Емпирија

Итеративни алгоритам

Хеуристика

Библиографија

- [1] Ian Millington; John Funge. In *Artificial intelligence for games second edition*. Morgan Kaufmann, Burlington, MA, 2009.
- [2] Mat Buckland. In *Programming Game AI by Example*. Wordware Publishing, Inc., Plano, TX, 2005.
- [3] Steve Woodcock. Flocking: A simple technique for simulating group behavior. In Mark DeLoura, editor, *Best of Game Programming Gems.*, pages 297{310. Course Technology, Cengage Learning, Boston, MA, 2008.
- [4] Steve Rabin. A* aesthetic optimizations. In Mark DeLoura, editor, *Game Programming Gems*, pages 264{271. Charles River Media, Boston, MA, 2008.
- [5] Johnson Geraint. Smoothing a navigation mesh path. In Steve Rabin, editor, *AI Game Programming Wisdom 3*, pages 129{140. Charles River Media, Boston, MA, 2006.
- [6] Pinter Marco. Realistic turning between waypoints. In Steve Rabin, editor, *AI Game Programming Wisdom*, pages 186{192. Charles River Media, Boston, MA, 2002.
- [7] A. Botea; M. Muller; J. Schaefer. Near-optimal hierarchical path finding. *Journal of Game Development*, 1, September 2006. URL <https://webdocs.cs.ualberta.ca/~mmueller/ps/hpastar.pdf>.
- [8] Chakrabarti. Heuristic search in restricted memory. pages 197{221. *Artificial Intelligence Journal (AIJ)*, Elsevier, 1989.
- [9] Xiao Cui; Hao Shi. A*-based path finding in modern computer games. *IJCSNS International Journal of Computer Science and Network Security*, 11, January 2011. URL http://paper.ijcsns.org/07_book/201101/20110119.pdf.