TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia Teaduskond

Tarkvarateaduse instituut

Alain Veeber 142457IAPB

2D platformeri läbimine A* algoritmi abil

Bakalaureusetöö

Juhendaja: Martin Verrev

MSc

Tallinn 2018

1 Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Alain Veeber

Kuupäev: 21.05.2018

2 Annotatsioon

Bakalaureusetöö eesmärgiks on uurida kuidas saab rakendada A* teeotsingu algoritmi 2D platformerile ning realiseerida see Unity mootoril kasutades uut *Tilemap* süsteemi. Töö tulemus on mõeldud Unity mängumootorit kasutavale arendajale, kes soovib realiseerida antud algoritmi oma mängus.

Töö käigus luuakse Unity mootoris *Tilemap* objekti kasutades hulk erinevaid 2D tasemeid. Pärast seda luuakse peamised mänguobjektid – mängija ja tasemete arhitektuur - mis on vajalikud algoritmi rakendamiseks. Selle tulemuseks on algoritmi märgistatud teekond mängija algpunktist lõpppunkti järgides balauresetöös kirjeldatud 2D platformeri reegleid.

Töö käigus kirjeldan mis on 2D platformer, võrdlen olemasolevaid teeotisngu algoritme, ning põhjendan A* algoritmi valiku. Kirjeldan üldsine mängutasemete ehitamise protsessi Unity mootoril ning realiseerin mängutasemed algoritmi rakendamiseks. A* algoritmi efektiivsuse valideerimiseks realiseerin selle loodud mängutasemetel ja annan hinnangu selle edukusele.

Tulemuseks on keskkond, mis lahendab taseme ja viib mängija vähima sammude arvuga taseme algusest lõpppunkti.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 43 leheküljel, 5 peatükki, 53 illustratsiooni, 5 tabelit.

3 Annotation (In English)

The purpose of the bachelor thesis is to investigate how the A * pathfinding algorithm can be applied to the 2D platformer and implement it on the Unity engine using the new Tilemap system. The result of the work is for a developer using the Unity game engine who wants to implement this algorithm in his game.
During the process, a set of different 2D platformer levels are created in the Unity engine using the new Tilemap object. After that, the main game objects the player and the architecture of the levels are created to implement the algorithm. This results in a path marked by the algorithm from the starting point of the player to the end point following the 2D platformer rules described in the thesis.
In the course of my work, I will describe what is a 2D platformer, compare existing pathfinding algorithms, and justify the A * algorithm selection. I describe the general process of creating game levels on the Unity engine and create the game levels themselves to implement an algorithm on them. To validate the efficiency of the A * algorithm, I use it at the levels created and give an assessment of its success.
The result is an environment that solves the level and leads the player through the shortest path to the end point of the level.

The thesis is written in Estonian and contains 43 pages of text, 5 chapters, 53 illustrations and 5 tables.

4 Lühendite ja mõistete sõnastik

2D Platformer	Mäng, milles kaamera näitab tegelast kõrvaltvaades.
2D platformeri tase	Keskkond, mida inimene näeb kõrvaltvaates ning kus toimub mängu tegevus
A* algoritm	Otsingualgoritm, mis leiab kõige optimaalsema teed kahe punkti vahel kasutades tippe
C#	Objektorienteritud programeerimiskeel, mida kasutab Unity mootor.
Collider	Füüsikaline objekt, mis registreerib kahe objekti vahel toimunud kokkupõrkeid.
Grid	Teljestikobjekt, mis jaotab keskkonna võrdseteks ruutudeks
Indie mäng	Väikeettevõte või ühe isiku poolt valmistatud mäng.
NPC	Non-player character. Arvuti poolt kontrollitavad tegelased.
Node	Graafi/maatriksi tipp
Pathfinder	Algoritm, mis aitab liikuda tegelasel punktist A punkti B, tihti kasutatakse labürintides.
Skript	C# koodifail, mida kasutab Unity mänguelement(näiteks liikumisskript, kokkupõrgeskript)
Sprite	Ühe elemendi kaadri/oleku kujutis
Sprite renderer	Seob objekti sprite’iga
Tile	Objekt/Blokk, millest koosneb Tilemap/TileBase
TileBase	Tilemap objekti kiht Unity mootoris
Tilemap	Maatrikskujuline blokkidest koosnev kaart
Unity mootor	Üks levinumaid mängumootoreid, märkimisväärne oma lihtsuse ning populaarsuse tõttu
UI	Kasutajaliides

Sisukord

5 Sissejuhatus

Bakalaureusetöö eesmärgiks on uurida, kuidas saab rakendada A* teeotsingu algoritmi 2D platformerile ning realiseerida see Unity mootoril kasutades uut *Tilemap* süsteemi. Töö raames uuritakse erinevaid rajaleidmise algoritme, valitakse neist parim ning rakendatakse see 2D tasemele, mis on valmistatud Unity mootori abil. Pärast seda arendatakse algoritm edasi, et seda saaks kasutada ka 2D platformeri laadses mängus. Töö tulemus on mõeldud Unity mängumootorit kasutavale arendajale, kes soovib realiseerida antud algoritmi oma 2D mängus.

5.1 Indie mängude kasv

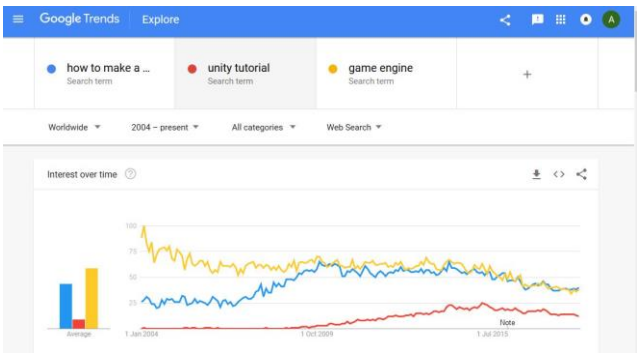
Indie mäng on mäng, mis on loodud väikeettevõte või ühe isiku poolt. Nende mängude arengut soodustas eelkõige mängumootorite teke (nagu Unity 3D) ning mänguarendamise õppematerjalide kättesaadavus. Lisaks sellele on kerkinud mobiilsed platvormid (IOS, Android), mis avasid veelgi rohkem võimalusi mängude müümisele. Uued mänguloomise võimalused ning uued mobiiliplatvormid soodustasid Indie mängude loomist, mis tõstsid huvi selle mängutüübi ning *Indie* mängude arendamise vastu.

Allpool on toodud Google'i graaf, mis näitab mängude loomisega seotud märksõnade populaarsust.

Sinine – how to make a game (Kuidas luua oma mängu)

Punane – unity tutorial (Unity mootori õppematerjal)

Kollane – game engine (Mängumootor)



Google näitab, et inimestel on huvi mängude loomise vastu. [16]

5.2 Eesmärgid

Enne projekti realisatsiooni uuritakse erinevaid otsingualgoritme kuna neid on mitu ning igalühel on olemas oma eelised ning puudused. Kuna töös kasutatakse A* algoritmi, tehakse selle algoritmi analüüs ning võrdlus teise algoritmiga.

Töö realisatsiooni osa algseks eesmärgiks on luua 2D platformeri keskkond kasutades Unity mootorit. Selleks kasutan uut *Tilemap* süsteemi, mis lihtsustab 2D tasemete loomist. Pärast seda loon mängija ning sihtmärgi objekti.

Töö järgmiseks eesmärgiks on rakendada taseme läbimise tippude asetamist, mida A* algoritm hakkab kasutama teekonna otsinguks. Kui tipud ja peamised objektid on olemas (mängija, sihtmärk), siis alustan A* algoritmi kirjutamist ning selle 2D platformeri kujule üleviimist.

Viimaseks eesmärgiks on luua erinevad tasemed, mis kontrollivad, et algoritm töötab õigesti.

5.3 Meetod

Kõigepealt valin otsingualgoritmi, võrdlen nende eeliseid ja puuduseid ning valin nendest parima. Pärast seda ehitan keskkonna koos tasemetega. Selleks valisin Unity 2D mootori kuna oman sellega kogemust ja see lihtsustab paljusid elemente - tasemete ehitamine, kokkupõrgete määramine - ning lubab keskenduda algoritmidele ja algoritmide testimisele. Pärast seda loon arvuti poolt juhtava tegelast (*NPC'd*) ning defineerin talle sellised juhtimise funktsioonid nagu "mine paremale/vasakule/ülese/alla". Valides rajaleidmise algoritmi (Antud projektis A* algoritm) ehitan seda ümber nii, et see töötaks 2D platformeril ning ühendan selle tegelase juhtimise funktsioonidega.

Lõpptulemuseks on testimiskeskkond, mis valideerib arvuti poolt juhitava tegelase rajaleidmise algoritmi erinevatel tasemetel.

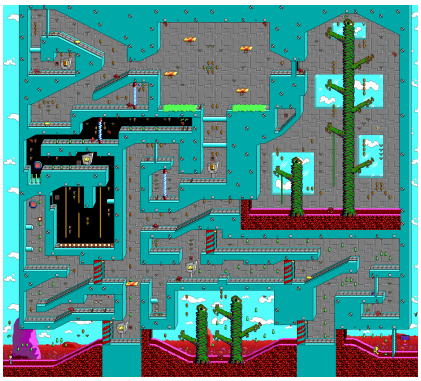
6 Taust

On olemas väga mitmeid 2D labürintide läbimise algoritme nagu "*Wall follower*", A*, "*Random mouse*" jne. Kahjuks ei ole võimalik antud algoritme ilma kohaldamata kasutada 2D platformeri laadses labürindis, kuna need on mõeldud 2D pealtvaates labürintide jaoks. Lisaks platformeri tegelasele lubatud liikumissuundadele (parem/vasak) on tarvis arvestada mitme lisaparameetriga - tegelast mõjutava gravitatsioonijõu, tema hüppe kaugust, hüppe kõrgust, taseme korruseid, kas tegelane suudaks hüppata järgmisele korusele või on vaja otsida teist teed.

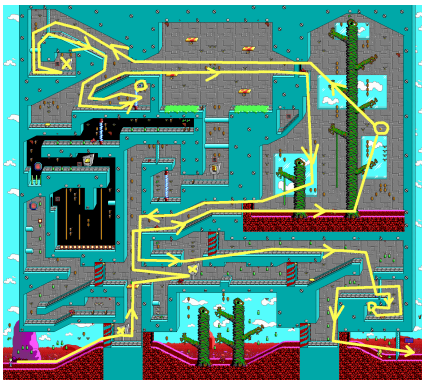
6.1 2D Platvormer

2D platformer on mängutüüp, milles kaamera näitab tegevust kõrvaltvaates. Reeglina on platformeri mängudes üks eesmärk: jõuda taseme algusest taseme lõppu. Selleks, et jõuda taseme lõpuni tuleb liikuda mööda platforme, mis kujutavad endast ühe suure labürinditaolise struktuuri korruseid, mida mängija peab läbima liikudes paremale/vasakule ning hüppates.

Mängu huvitavaks muutmiseks lisatakse tasemetele erinevaid elemente, mis teevad selle läbimise raskemaks. Näiteks saab lisada lukustatud uksi või liikuvad platvormid, mille avamiseks/aktiveerimiseks on vaja leida sobivat võtit või lülitit. Samuti lisatakse tasemetele vastaseid, kes nähes peategelast, üritavad teda takistada tulistades/puutudes teda. On olemas ka tavalised lõksud, mida peategelane ei tohiks puutuda või lülitid, mis lülitavad lõksu välja. Kõiki eelmainitud takistusi läbimist aitavab igasugused boonused, näiteks relvakuulid, eluboonused ning lihtsalt punktiboonused. Klassikalise 2D platformeri näide koos taseme elementidega on toodud lisades.



"Commander Keen 6- Aliens Ate my Baby Sitter" tase nr. 10 [8]



Teekond algusest lõpuni kasutades minimaalset hulka võtmeid [8]

6.2 Mis on Unity?

Unity on mitme-platvormiline mängumootor, mida arendab Unity Technologies. Algne versioon tuli välja 2005. aastal ning sellest ajast on välja tulnud 6 peamist versiooni. Mootor on mõeldud 2D ja 3D mängude loomiseks ning suudab neid väljastada 27-le erinevale platvormile. 2014. aastal pälvis Unity mootor Suurbritannia "Parima Mootori" auhinda Develop Industry Excellence autasustamise tseremoonial. [3]

Kuulsamad Unity mootoriga tehtud mängud on: Kerbal Space Program ja Pokemon Go. [2]

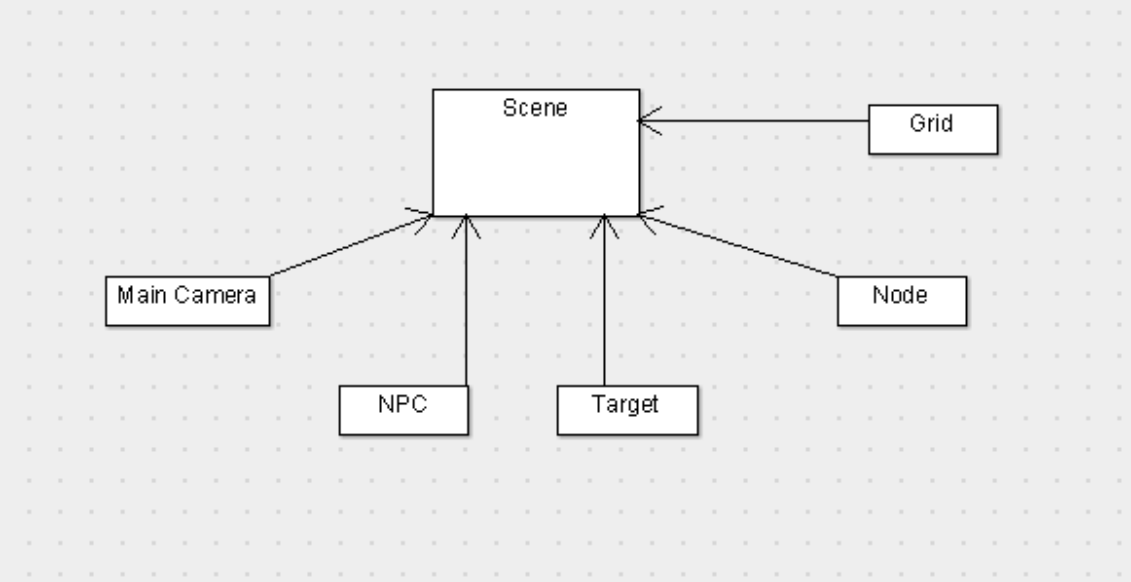
Antud projekti jaoks on valitud just see mootor kuna see suudab lihtsustada mitmeid mänguelementide loomise protsesse tänu oma lihtsale *UI*'le, mis toetab *Drag&Drop* funktsionaalsust. Samuti antud mootoril on olemas ulatuslik dokumentatsioon ning mitmeid õppevideod erinevatel teemadel. [2]

6.2.1 Unity objektid ja nende moodulid

Selles peatükis kirjeldan Unity peamisi objekte ning mooduleid, et lugejal tekkiks ettekujutus projekti struktuurist enne kui ta jõuab projekti realisatsioonini. Kõike, mis toimub mängus , näeb Unity mootor objektidena. Unity objektid kujutavad endast punkti, milles on salvestatud selle asukoht, suurus ning suund. Need objektid sisaldavad tihtipeale igasuguseid lisamooduleid, mis iseloomustavad nende objektide käitumist. Kõige tavalisemad moodulid on *Sprite renderer*, *Collider*, *RigidBody2D* ja skriptid. [9]

6.2.1.1 Scene(Stseen)

Unity tasemeid salvestatakse stseenina. Stseen sisaldab kõiki objekte. Unity *Drag&Drop* süsteem lubab asetada objekte lohistades neid ekraanile.



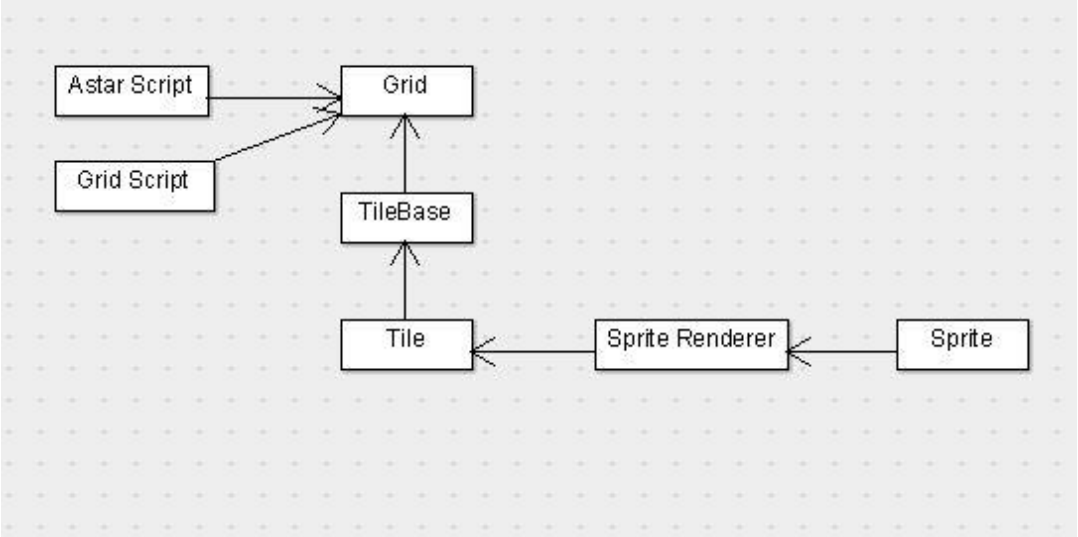
Scene'i objekti struktuur

6.2.1.2 Main Camera(Peamine kaamera)

Kaamera objekt, mis lubab kasutajal näha keskkonnas toimuvat.

6.2.1.3 Grid objekt

Grid on keskkonnaobjekt, mis jaotab taseme maatriksiks. See objekt tekib uue *Tilemap'i* loomisel automaatselt. Keskkonna käivitamisel hakkavad tööle selle objektis asuvad *Grid Script* (asetab tipuobjekte maatriksile) ning *Astar Script* (otsingualgoritm). Sisaldab 2 *TileBase* objekte, kus asuvad kõik blokid. Üks on takistustega (platvormiblokkidega) *TileBase* ning teine on taustablokkidega *TileBase*



Grid objekti struktuur

6.2.1.4 *TileBase objekt*

Kiht, kus asuvad kõik *Tile* blokid. *Gridi* skript võtab *TileBase* koos Platvormiblokkidega ning skaneerib seda läbi. Platvormiblokkidega *TileBase* omab *TileBase colliderit*, mis hõlmab kõiki blokke. Kui *NPC* astub Platvormibloki sisse, siis tuleb konsooli teade, et *NPC* liikus seina sisse.

6.2.1.5 *Tile objekt*

Tile objekt on komponent, millest ehitatakse *TileBase* kiht. Selle sees asub ainult *Sprite renderer*, mis sisaldab seina *sprite'i*

6.2.1.6 *Platvormiblokk*

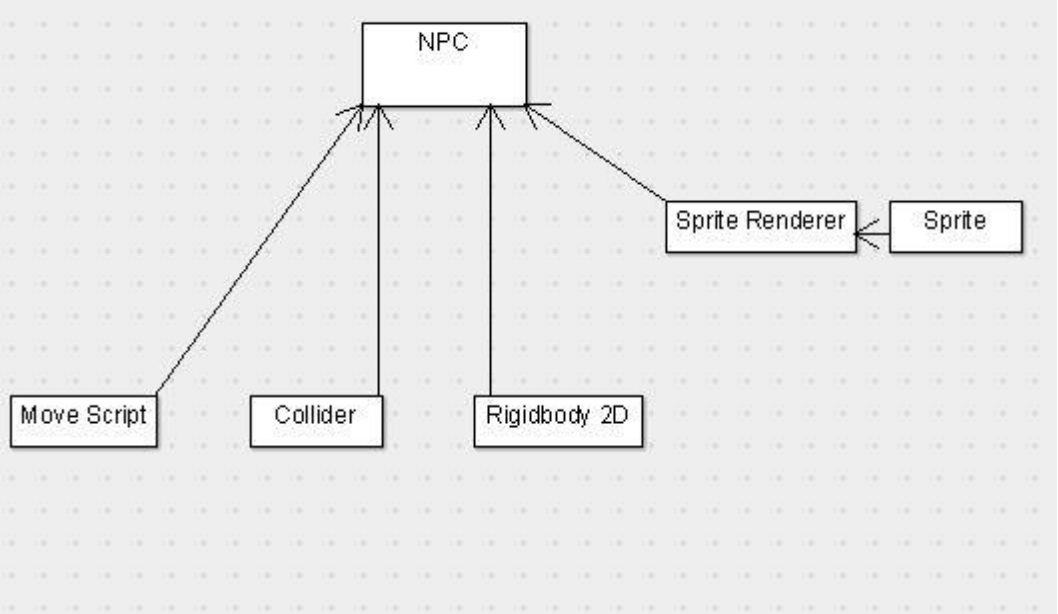
Platvormiblokk kujutab endast tavalist *Gridi* ühiku suurust bloki. Kui skaneeritakse kogu tase ja luuakse 2D maatriks algoritmi jaoks, siis neid blokke ignoreeritakse kuna nende sees ei tohi käia. Antud blokk on salvestatud *Tile* objektina, et neid saaks kergesti lisada koordinaatteljele.

6.2.1.7 Taustablokk

Taustablokk on lihtsalt taustaks seatud blokk.

6.2.1.8 NPC valmistamine

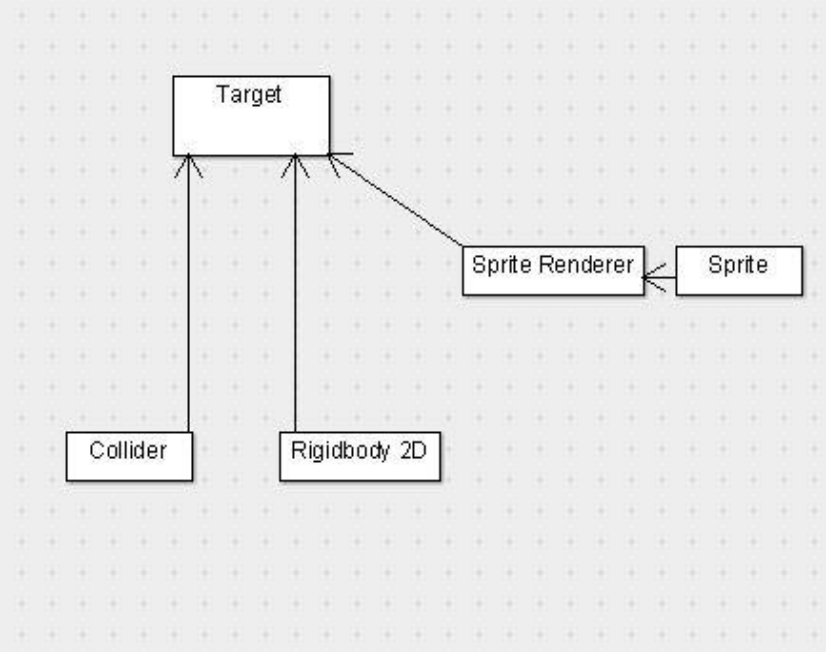
NPC on arvuti poolt juhitav objekt, mida algoritm liigutab sihtmärgi poole. Antud kontekstis see on sprite, millel on olemas *Collider* ja *RigidBody2D* (et algoritmi tipud teaksid, kus ta asub) ning millele on lisatud skript, mis vastutab tema liikumise eest. *Collider* lubab maatriksi tippudel registreerida kokkupuudet *NPC'ga* ning edastada oma kordinaadid A* algoritmile. *Rigidbody2D* annad objektile füüsilist keha, et ta ei läheks läbi teisi objekte. *NPC* skript sisaldab liikumise funktsioone, mida A* algoritm välja kutsuma.



NPC objekti struktuur

6.2.1.9 Sihtmärk

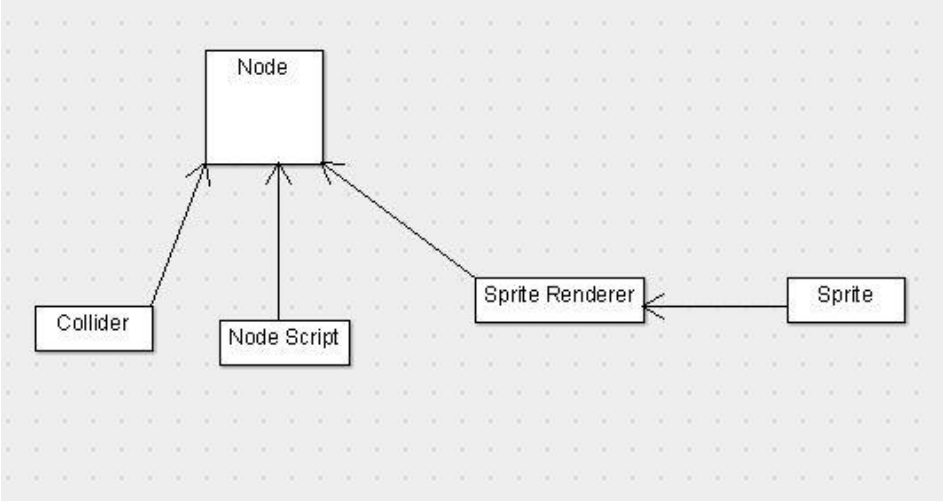
Sihtmärgiks kasutatakse sarnast objekti nagu *NPC* (tal on oma *sprite*, *collider*, *RigidBody2D*), ainult tal puudub liikumise eest vastutav skript. Sihtmärgi asukoht samuti registreerib tipp, millel ta asub ning saavab oma koordinaadid *pathfinding* algoritmile.



Sihtmärgi struktuur

6.2.1.10 Tipuobjekt

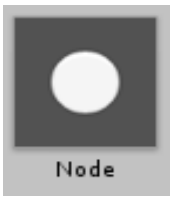
Pärast taseme skaneerimist lisatakse läbitavatele blokkidele tipud, mis tekkivad blokkide keskele. Tippudel on olemas colliderid, mis registreerivad, kas nende kohal on olemas *NPC* või sihtmärk. Samas on nendele lisatud skript, mis sätib nendele kordinaadid, tipu vanemat ja tipu kaalud jne. Lisaks skript sisaldab *collideri* funktsiooni, mis registreerib, mis objekt asub nende kohal (*NPC* või sihtmärk) ning saadab need andmed A* algoritmile.



Tipuobjekti struktuur

6.2.1.11 Objekti sprite'ide tabel

Siia on lisatud peamiste objektide *sprite'id*, et lugejal oleks kergem orienteeruda projekti testimiskeskkonnas.

	Platvormibloki sprite
	Taustabloki sprite
	NPC sprite
	Sihtmärgi sprite
	Tipuobjekti sprite (Võetud Unity sisseehitatud ressursidest)

6.2.1.12 Collider

Unity mootoris objektidevahelist käitumist lubavad nende *colliderid*. *Collider* on objekti moodul, mis registreerib 2 *collideri* vahelist kokkupuudet. Antud projektis colliderid leiduvad *NPC'l*, sihtmärgil, tipuobjektil, ning *TileBase*. Tipuobjekt registreerib kui teda puudutab *NPC* või sihtmärk ning saadab teadet koos oma asukohaga A* algoritmi skriptile. Kui *NPC* astub seina sisse, siis *TileBase collider* kirjutab konsooli teadet, et algoritm töötas valesti.

6.2.1.13 RigidBody 2D

RigidBody2D on moodul, mis annab kehale füüsilist keha. Seda moodulit omavad *NPC* ja Sihtmärgi objektid ning nende ainuke roll on lasta tipuobjektidel registreerida nendega kokkupõrkeid.

6.2.1.14 Sprite Renderer

Objekti moodul, mis laseb objektidel omada visaalset pilti. Sisaldab objekti *sprite'i*.

6.2.1.15 Unity skriptid

Unity mootori skriptid sisaldavad objektiga seotud koodi, mis on kirjutatud ainult C# keeles. Skripti loomisel tekivad lisaks klassile ka sisseehitatud funktsioonid nagu *Start()* (Kirjeldavad, mida skript teeb selle initsieerimisel) ning *Update()* (Kirjeldab, mida skript teeb iga mängu miinimumajaühiku tagant). [9]

7 Algoritmide analüüs ja valik

Mängudes võib tihtipeale kohata olukorda, kus mängija vajutab hiirega mingile kohale ning mängutegelane läheb sinnasamasse. Sellist käitumist võimaldab *pathfindingu* algoritm, mis suunab mängutegeast hiirega märgitud kohale. Otsingualgoritme oma kõige lihtsamal kujul rakendatakse maatriksikujulises labürindis, kus näiteks number 0 on sein, 1 on põrand, 2 on algus ning 3 on lõpp.

```
1111111111          1111111111
1010000311          1010000311
1010111101          1010111101
1010010001          1010010001
1001010111          1001010111
1100000001          1100000001
1111111121          1111111121
```

Maatrikslabürint ning selle lahendus, punased nullid näitavad teekonda.

Alltoodud tabelis on näidatud mitmeid otsingualgoritme ning neid iseloomustavad omadused.

Tulemuste veerg kirjeldab algoritmi tulemust. „1“ tähendab, et algoritm annab ainult üht tulemust (Viimane algoritm on A* algoritm ning annab üht kõige lühimat tulemust). „Kõik“ tähendab, ei algoritm leiab kõik võimalikud teekonnad ning „Kõik+“ leiab kõik võimalikud teekonnad, mis sisaldavad ebavajalikke teekondi. Kui parameeter sisaldab sõna „Lühim“, siis tulemuste seas on olemas parim võimalik teekond.

Garantii veerg ütleb kas algoritm garanteerib teekonna leidmist (kui see eksisteerib). Mõned algoritmid ei pruugi leida tulemust erinevatel põhjustel. Näiteks „*Random mouse*“ võib võtta kuni lõpmatuseni aega , et leida tee sihtmärgini suurtes labürintides kuna ta pidevalt valib täiesti suvalist teed.

Fookus atribuut ütleb, kas algoritm keskendub ainult mängjale („Sina“), mängija ümbrusele kasutades otsinguks tippe („Sina+“) nagu dijkstra ning A* algoritmi puhul või labürindile (Keskkond). Viimase puhul algoritmid leiavad kõik valed teekonnad ning ei luba mängijal sinna minna.

Inimese poolt lahendatav parameeter ütleb, kas inimene suudab lahendada seda algoritmi mängija vaatepunktist („Seest“) või ülevalt vaadates („Ülevalt“). Raskemad algoritmid ei ole inimese poolt tehtavad ning vajavad arvuti arvutamisvõimet.

Ei nõua koridore parameeter ütleb kas antud algoritmi saaks kasutada ruumidega labürindis. Näiteks selline algoritm nagu *„Wall Follower“* ei hakka tööle kui mängija asub ruumi keskel kuna ta nõuab seina, mida ta peab järgima (sama lugu ruumi keskel asuva sihtmärgiga).

Mäluta atribuut ütleb, kas algoritm ei nõua andmete hoidmiseks mälu. Näiteks A* algoritm peab meeles pidama tipud, mida ta läbima peab.

Kiire atribuut ütleb, kas algoritm on piisavalt kiire. Antud kontekstis peaksid algoritmid läbima igat tipu ainult üks kord ning algoritmi keerukus ei tohi ületada O(n^2), kus n on labürindi külje tippude arv. A* algoritm on kiire algoritm (keerukus O(lgn)) kuna ta soosib oma otsingus tippe, mis on sihtmärgile lähemal.

Algoritmi nimi	**Tulemus**	**Garantii**	**Fookus**	**Inimese poolt lahendatav**	**Ei nõua koridore**	**Mäluta?**	**Kiire**
Random Mouse	1	Ei	Otsija	Seest/Ülevalt	Ei	Jah	Ei
Wall Follower	1	Ei	Otsija	Seest/Ülevalt	Jah	Jah	Jah
Pledge Algorithm	1	Ei	Otsija	Seest/Ülevalt	Jah	Jah	Jah
Chain Algorithm	1	Jah	Otsija+	Ei	Jah	Ei	Jah
Recursive Backtracker	1	Jah	Otsija	Ei	Jah	Ei	Jah
Tremaux’s Algorthm	1	Jah	Otsija	Seest/Ülevalt	Ei	Ei	Jah
Dead End Filler	Kõik+	Ei	Kesk-kond	Ülevalt	Ei	Jah	Jah
Cul-de-sac Filler	Kõik+	Ei	Kesk-kond	Ülevalt	Ei	Jah	Jah
Blind Alley Sealer	Kõik+	Jah	Kesk-kond	Ei	Ei	Ei	Jah
Blind Alley Filler	Kõik	Jah	Kesk-kond	Ülevalt	Ei	Jah	Ei

Collision Solver	Kõik/ Lühim	Jah	Otsija+	Ei	Ei	Ei	Jah
Shortest Paths Finder (Dijkstra)	Kõik/ Lühim	Jah	Otsija+	Ei	Jah	Ei	Jah
Shortest Path Finder (A*)	1 Lühim	Jah	Otsija+	Ei	Jah	Ei	Jah

Tabel koos erinevate pathfindingu algoritmidega [6]

7.1.1 Algoritmide valik võrdluseks

Eeltoodud tabelist ma võtan algoritmide võrduseks algoritme vastavalt nende iseloomujoontele, mis on tähtsad pathfindingu algoritmi lahendamisel. Alustuseks algoritm peab garanteerima tulemust. Algoritmide tablis on olemas 5 algoritmi, mis ei vasta antud kriteeriumile ning neid võetaks võrdlusest välja. Järgmiseks kriteeriumiseks on algoritmi võimalus leida kõige lühimat teed. Sellele kriteeriumile vastavad ainult 3 algoritmi (*„Collision Solver“*, Dijkstra, A*). Antud juhul on need algoritmid parimateks kandidaatideks, kuid kahjuks peab võtma ära ka *„Collision Solver“* algoritmide võrdlusest. Põhjuseks on temale iseloomilik nõue, et seda algoritmi saab kasutada ainult ruumideta labürindis. Kui *„Collision Solver“* algoritmi kasutada labürindis, kus sithmärk asub ruumi keskel, siis algoritm ei leia seda. Kokkuvõttes on võrdluseks alles jäänud vaid Dijkstra ning A* algoritm.

7.1.1.1 Dijkstra algoritm

Dijkstra algoritmi saab endale ette kujutada valades labürindi algpunkti vett niikaua kuni see jõuab labürindi lahenduseni. Juhul kui on olemas raskemini kättesaadud paigad, siis algoritm peatub sel kohal ning uurib enne neid kohti, mis on kergemini uuritavad. Antud algoritm vajab mälu, mis hoiab meeles potentsiaalsed tipud ning see annab kõik võimalikud lahendused mille hulgas on olemas ka lühim teekond.

7.1.1.2 A algoritm*

A* algoritm on dijkstra algoritmi edasiareng ning talle kehtivad samad omadused. Lisaks sellele üritab algorim liikuda juba tuntud lahenduse poole, kuna ta teab juba enne kus asub sihtmärk ning on enne lahendamist seadnud igale paigale märgistust, mis aitavad algortitmil navigeerida. Antud algoritm vajab mälu, mis hoiab meeles potentsiaalsed tipud, ning see annab üht lahendust, mis on kõige lühem tee lahenduseni. [6]

Algoritmi nimi	**Tulemus**	**Garantii**	**Fookus**	**Inimese poolt lahendatav**	**Ei nõua koridore**	**Mäluta?**	**Kiire**
Shortest Paths Finder (Dijkstra)	Kõik/ Lühim	Jah	Otsija+	Ei	Jah	Ei	Jah
Shortest Path Finder (A*)	1 Lühim	Jah	Otsija+	Ei	Jah	Ei	Jah

7.1.2 Algoritmide võrdlus

Eeltoodud algoritme võrreldes on näha , et A* on tugeva eelisega. Ta teab juba enne, kus asub sihtmärk ning on märgistanud igat tippu, mis aitab tal kergemini leida sihtmärgi. Lisaks sellele A* algoritm leiab kohe ühte lühimat teekonda, samal ajal kui Dijkstra algoritm otsib kõik võimalikud teekonnad. Dijkstra algoritmi on parem kasutada olukordades, kui labürindis on vaja leida mutu sihtmärki. Lisaks sellele Dijkstra algoritmi kasutatakse suuremõõtmelisel navigeerimissüsteemides (nagu Google Maps). Seal võetakse kõik tipud ning töödeldakse läbi luues graafi, mis arvestab mitte ainult teekonna pikkust, vaid ka tee läbitavust, aega ning muid parameetreid[17]. Antud projekti raames on tähtis ainult teepikkus ning seetõttu on A* parem valik.

8 Realisatsioon

8.1 Tasemete ehitamine

Selleks, et testida algoritmi töökust on vaja ehitada testimiseks tasemeid. Selleks sobib hästi Unity 2D mootori keskkond kuna see aitab lihtsustada paljusid elemente. Tasemed on maatrikskujul ning platvormid koosneb ruudukujulistest blokkidest. Lihtsamaks ehitamiseks on vaja ette valmistada keskkond, et blokid sobiksid kokku maatriksi sektoriga.

8.1.1 Unity Tilemapi loomine

Alates Unity 2017.2 versioonist on mootorile lisatud *Tilemap* objekt. *Tilemap* kujutab endast maatriksi, mida saab täita blokkidega nii, et nad saaksid üksteise külge kleepuda, mis on väga vajalik 2D mängude disainimisel. *Tilemapi* plussiks on see, et iga blokki saab seostada maatriksi koordinaatidega ning see et blokkidele saab automaatselt sättida *colliderid*, mis minimiseerivad vigade tekkimist (kui *colliderid* on käsitsi pandud, siis võib tekkida olukord, kus objekt jääb kahe *collideri* vahele). [14] *Tilemapi* loomisel tekivad stseenile grid objekt, mis jaotab stseeni maatriksiks ning *Tilebase* objekt, kuhu lisatakse *Tile* objekte.

8.1.2 Tile Palette tööriist

Selleks, et *TileBase'ile* lisada *Tile* objekte, on olemas Unity mootoril selline töö riist nagu *Tile Palette*. Sinna on vaja lisada kõiki joonistamiseks vajavad *Tile* objektid. Et joonistada *TileBase'i* on vaja valida palettist *Tile'i* ning vajutada stseenile, et seda sinna asetada.

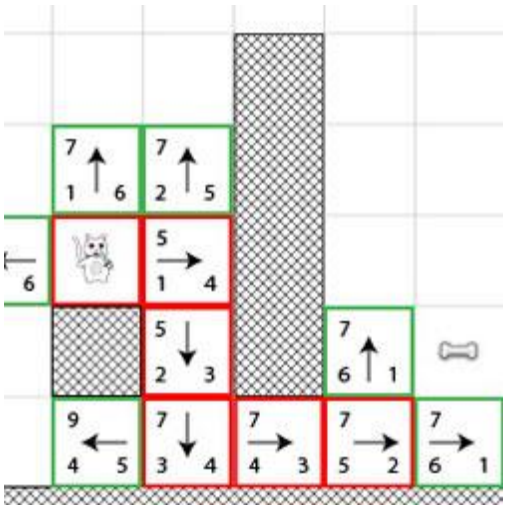
8.2 A* algoritm

A* (A star) algoritm on üks graafi läbimise algoritme, mida tihti kasutatakse otsingutes. Selle algoritmi autoriteks on Peter Hart, Nils Nilsson, Bertram Raphael Stanfordi Uuringu Instituudis 1968. aastal ning kujutab endast arendatud Dijkstra algoritmi.

A* kasutab otsinguks tippe, mis kujutavad endast graafi koordinaati. Oma sihtpunkti otsides, üritab algoritm läbida kõik võimalikud teed (alustades kõige odavamatest) ning tagastada kõige lühem/odavam tee. Igal tippul on olemas oma X ja Y koordinaat, oletatav kaugus (Manhattani kaugus) sihtpunktini, läbimise maksumus/kaal, kogukaal (oletatav kaugus+ kaal) ning tipp, kust otsing tuli (tipu vanem). [10][11][12]

8.2.1 Kuidas A* algoritm töötab?

Alguses algoritm võtab alguspunkti ning lisab seda avatud andmetekogumisse (edaspidi listi), pärast seda algoritm otsib selle tipu kõiki naabreid, mida ta pole veel külastanud ning lisab need avatud listi. Pärast seda ta valib avatud listist kõige odavama kogukaaluga tipu ning jätkab oma otsingut seal kuni ta jõuab sihtpunktini. Kui otsingu ajal tuleb välja, et mingi tipu naabrid on kõik läbitud, siis lisatakse see tipp suletud listi, kus asuvad tipu, mida algoritm on jõudnud täielikult läbi uurida. Kui sihtpunkti pole või selleni on võimatu jõuda, siis tulemuseks ei saavuta algoritm teekonda ning kõik tipud asuvad suletud listis. [10][11][12]



A* algoritmi tööprotsess, alumine parem number näitab Manhattani kaugust, alumine vasak näitab kaalu, mis koosneb läbitud tippude kaalude summat, ülemine number näitab tipu kogukaalu, mille järgi ta valibki järgmist tippu.[12]

8.2.2 Manhattani kaugus

Et A* algoritm teaks, mis suunas ta peaks liikuma, peab ta lisaks tippude kaalule arvestama heuristilist kaugust sihtpunktini. Antud projektis kasutatakse Manhattani kaugust, mille nimi tuleneb New-Yorgi Manhattani linnaosa maatrikskujulisest majapaigutusest. [5]

Arvutatakse järgmiselt:

$$s = |\,Xalgus - Xlõpp\,| + |\,Yalgus - Ylõpp\,|$$

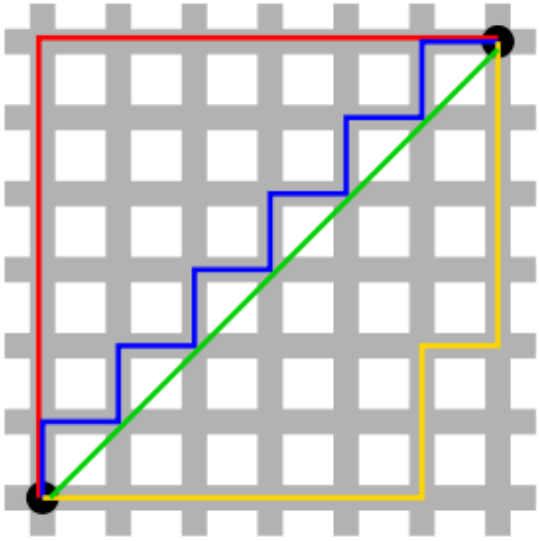
s – Manhattani kaugus

Xalgus - Alguspunkti X-koordinaat

Yalgus - Alguspunkti Y-koordinaat

Xlõpp - Lõpupunkti X-koordinaat

Ylõpp - Lõpupunkti Y-koordinaat



Manhattani kaugused [8]

Ülemisel pildil on illustreeritud Manhattani kaugused. Punane, kollane ja sinine teekonnad on erinevad, kuid nende Manhattani kaugus on sama - 12 ühikut. Roheline joon on kahe punkti vaheline „tegelik“ kaugus, kuid seda antud projektis ei kasutata, et vältida murdarvde kasutamist.

8.3 Tippude lisamine

Tipude lisamine antud projektis toimub järgmiselt:

1) Vaadatakse taseme maatriksi pikkust ja kõrgust ning luuakse vastav 2D massiiv, kuhu pannakse tipud.
2) Taseme skript võtab Tilemapi objekti, mis sisaldab endas kõiki platvormiblokke ning läbib kõgu taseme maatriksi.
3) Kui antud koordinaadil puudub blokk, siis sinna keskele luuakse tipuobjekt ning seadistatakse sinna objekti atribuudid (x-koordinaat, y-koordinaat, kaal). Antud tipp lisatakse 2D massiivi.
4) Kui antud koordinaadil on olemas blokk, siis minnakse edasi.
5) Maatriksi lõpuni jõudes, teavitavad tipuobjektide skriptid A* algoritmi skriptile, et *NPC'i* ning sihtmärgi koordinaadid on teada, ning A* algoritm saab oma tööd alustada

8.4 A* algoritmi üleviimine 2D platformeri kujule

Pathfindingu algoritmi 2D platformerile üleviimine toob kaasa mitmeid muutusi algoritmis. Maatrikslabürindi peaelementideks on algus, sein, põrand ja lõpp ning mängija saab liikuda üles, alla, paremale ja vasakule. 2D platformeri elementideks on samuti sein, põrand, algus ja lõpp, kuid peaerinevus seisneb mängija liikumises- ta hüppab platvormilt platvormile ning talle rakendub gravitatsioonijõud (arvestades seda, et mänguja ei saa lennata). Järelikult peab lubama A* algoritmile arvestada hüppeid.

Antud projektis on olemas piirangud, mis on iseloomulikus 2D platformeri mängudele ning mida algoritm peab arvestama:

1) Hüppekõrgus- määrab kui kõrgele mängija hüppab, realiseeritakse hüppeväärtustega, mis on kirjeldatud hiljem.
2) NPC saab käia ainult platvormidel (s.t. tema asukoha all peaks asuma blokk, mille peal ta käib)
3) NPC ei saa liikuda ülespoole kui hüppamisel tema ülesliikumist takistab lagi
4) NPC liigub allapoole kui tema all puudub blokk, ning tal puuduvad hüppeväärtused.

8.4.1 Nõuded

Nõue	**Valideerimine**
Keskkonna käivitamisel tekivad sama koodiga alati õiged tipud.	Sama kood suudab luua algoritmi jaoks õigeid tippe isegi siis kui tasemed on erinevad
Algoritm töötab sihtmärgi asukoha muutmisel.	Kui *NPC* leiab sihtmärgi, siis ta leiab seda ka siis kui samal tasemel muuta sihtmärgi asukoha .
Algoritmi otsingule on rakendatud hüppamise simulatsioon.	Kui sihtmärk on hüppamiskaugusest väljas, siis algoritm läheb mööda teist teed.
Mängija ei saa kätte sihtmärgi kui see on kättesaamatus kohas	Kui teekond sihtmärgini puudub (n. see asub seinte taga), siis tuleb teade, et lahendus puudub.
Algoritm töötab kõikidel tasemetel	*NPC* jõuab sihtmärgini igal tasemel

8.4.2 Hüpete arvutused

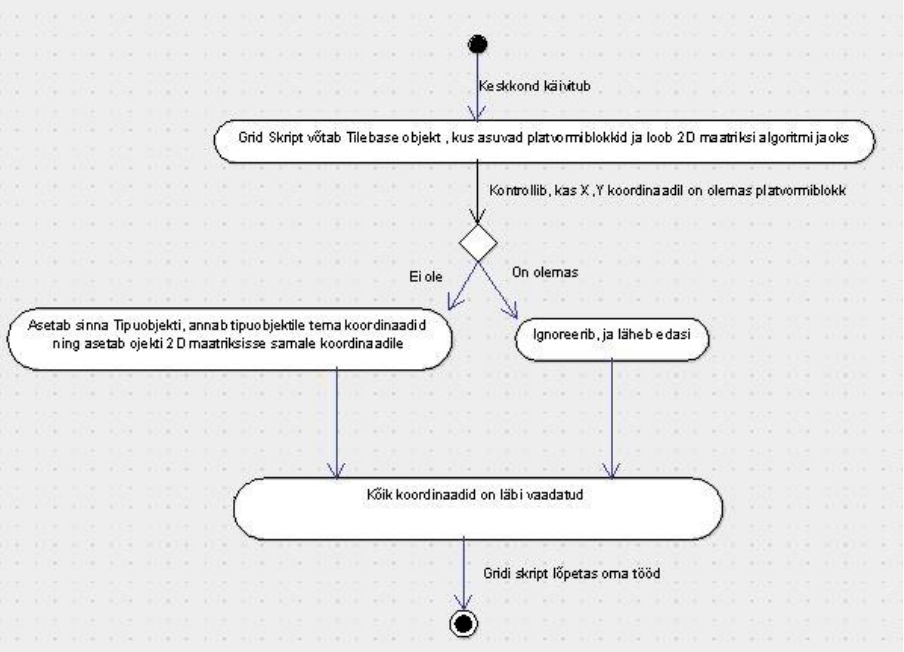
Selleks, et algoritm arvestaks platvormivahelisi hüppeid on vaja lisada algoritmisse muudatusi, mis simuleeriks hüppeid. Need on kirjeldatud koodis hüppeväärtustena. Olgu *NPC* hüppekõrguseks arv 2. See arv näitab, et *NPC* suudab ülesse hüppamisel liikuda maksimaalselt 2 blokki ülespoole ning iga bloki jõudmiseni suudab ta liikuda kas paremale või vasakule. Alla kukkumisel võib tegelane samuti liikuda iga bloki tagant paremale või vasakule. [13] Lubatud hüpete näidised asuvad lisades.

8.5 Testimiskeskkond

Testimiskeskkond käivitatakse Unity mootori abil. Antud peatükis kirjeldatakse, millised skriptid töötavad testimiskeskkonna käivitamisel ning, mida nad teevad.

8.5.1 Grid Script (Maatriksi skript)

Testimiskeskkonna käivitamisel tekkivad kõik objektid stseenile ning käivituvad nende skriptid. Eelkõige käivitub *Grid* objekti sees asuv skript, mis asetab tipub *Tilemapile*. Ta võtab *TileBase* objekti, kus asuvad kõik platvormiblokkid ning hakkab järjest kontrollima, kas sel koordinaadil on olemas blokk või mitte. Kui blokk on olemas, siis skript läheb edasi. Kui ei ole, siis sinna asetatakse Tipuobjekt. Tipuobjekti loomisel käivitub, tipu skript, mis lubab tal salvestada enda koordinaadid ja kaalud. *Grid'i* skript annab tipule tema koordinaadid ning lisab tipu 2D massiivi, mida hiljem hakkab kasutama A* algoritm.

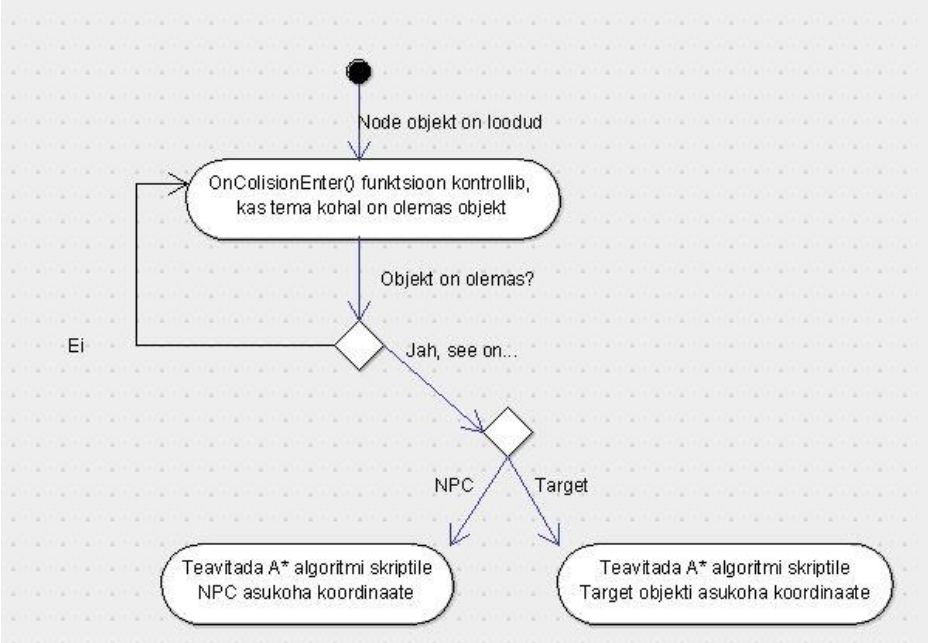


Grid Script tegevusdiagramm

Testimiskeskkonna käivitamisel läheb tööle eelkõige *Grid* objekti skript. Skript võtab platvormiblokkidega *TileBase* objekti ning hakkab järjest kontrollima, kas X ja Y koordinaadil on olemas blokk. Kui blokki ei ole, siis ta asetab sinna tipuobjekti. Kui kõik koordinaadid on läbi vaadatud, siis *Gridi* skript lõpetab oma tööd.

8.5.2 Node script (Tipu skript)

Tipu skript omab kaks peamist funktsiooni : salvestada oma parameetreid (koordinaadid, Manhattani kaugus, kaal) ning teavitada A* skriptile, et tema kohal asub objekt (*NPC* või sihtmärk). Kui tasemel puudub sihtmärk, siis tipule vajutades saab tekitada uut.



Node Script tegevusdiagramm

Gridi skripti töö tulemusena asetatakse tasemele tipuobjektid, mis kontrollivad kas nende kohal on olemas *NPC* või sihtmärk (Target). Kui on, siis tipuobjekti skript teavitab A* algoritmi skriptile, et tema kohal on olemas objekt.

8.5.3 Target script (Sihtmärgi skript)

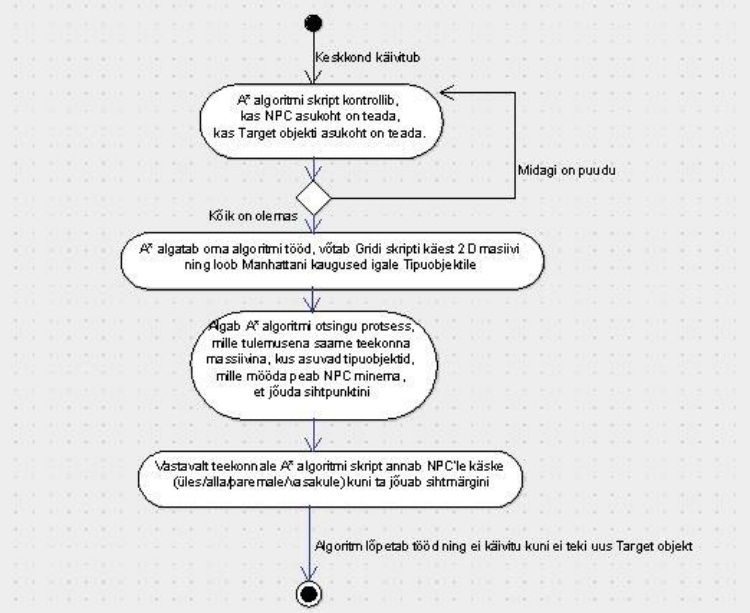
Sihtmärgi skript hävitab sihtmärgi, kui see puutub kokku *NPC collideriga*.

8.5.4 NPC script (mängija liikumise skript)

Omab liikumisfunktsioone (mine üles/alla/paremale/vasakule), mida kutsub välja A* algoritmi skript.

8.5.5 Astar script (A* algoritmi skript)

Eelkõige skript kontrollib, et tal on olemas nii sihtmärgi kui ka *NPC* asukoha koordinaadid. Kui neend on olemas, siis ta võtab 2D maatriksi koos tipuobjektidega ning salvestab neile nende Manhattani kaugused. Pärast seda toimub A* teeotsingu protsess, mille tulemusena tekib tippudest massiiv, mida *NPC* peaks järgima. Kui *NPC* jõuab tipu kohale, algoritm lõetab oma tööd, kuni tekkib uus sihtmärk.



A algoritmi tegevusdiagramm*

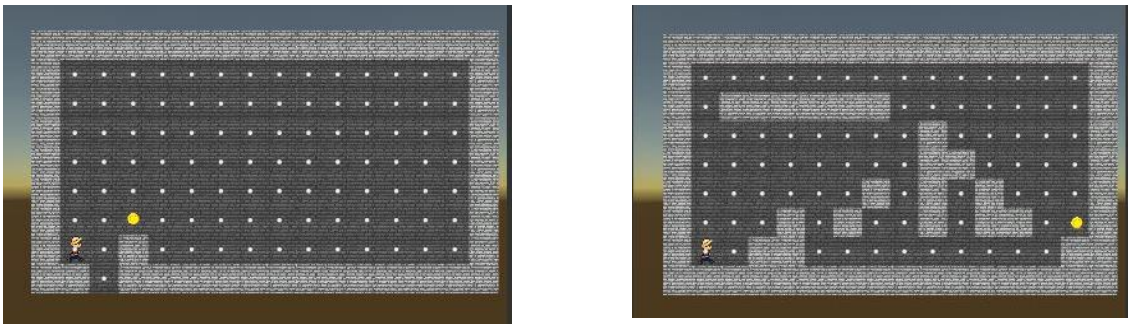
Pärast testimiskeskkonna käivitamist hakkab A* kontrollima, kas ta teab *NPC* ning sihtmärgi (*Target*) objektide asukohta niikaua kuni ta neid saab. Kui asukohad on olemas, siis ta võtab gridi skripti poolt vasmis saadud 2D massiivi koos tipuobjektidega ning sättib igalühele tipuobjektile tema Manhttani kaugust. Siis algab peamine A* otsinguprotsess, mis on eelnevalt kirjeldatud. Erinevus seisneb selles, et projekti algoritm ei luba *NPC* liikuda üle kahe bloki ülespoole (hüppetega seotud piirang). Kui rada sihtmärgini on leitud, siis algoritm märgistab tekkinud teekonda ning liigutab *NPC'd* sihtmärgi poole. Kui *NPC* puutub kokku sihtmärgiga, siis sihtmärk kaob ning algoritm lõpetab oma tööd.

8.6 Tulemused ja nõuete täitmiste näited

Tulemusena sai testimiskekond, kus *NPC* kasutades A* algoritmi leiab teed sihtmärgini. Järgmistes peatükkides on kirja pandud eelnevalt sätitud nõuded ning on toodud konkreetsed näited nende täitmise kohta.

8.6.1 Tippude tekkimine

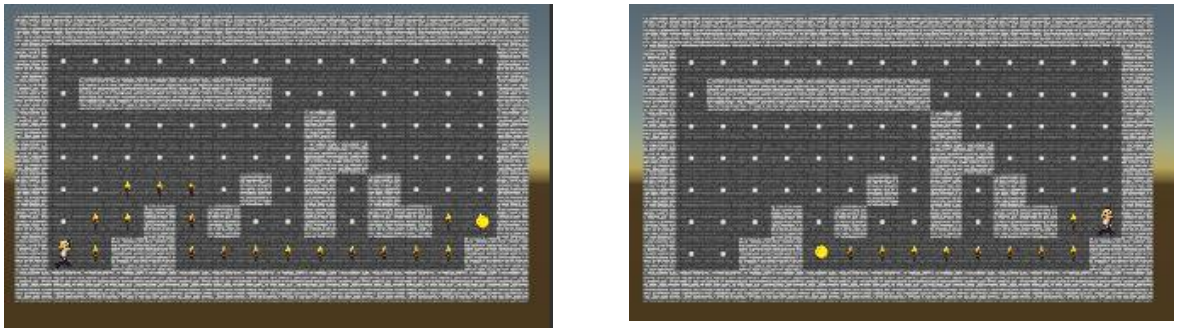
Algoritm suudab luua õigeid tipuobjekte taseme struktuuri muutmisel



Näidis tippude tekkimisest

8.6.2 Sihtmärgi muutmine

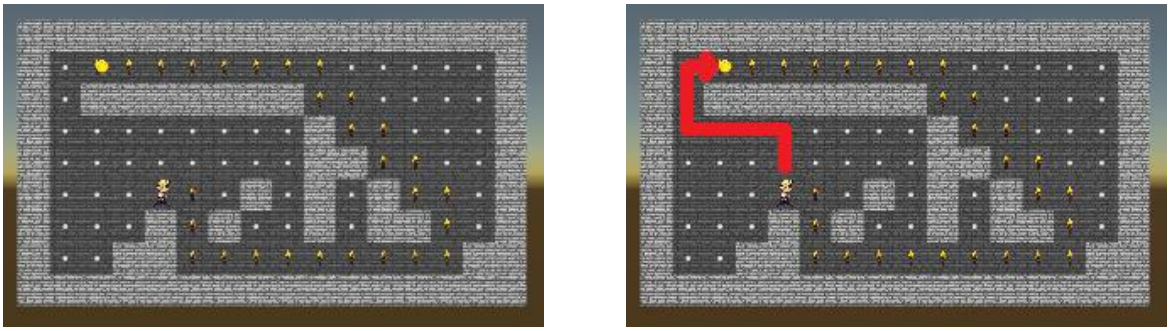
Kui muuta sihtmärgi asukoha, siis algoritm leiab ikkagi teed sihtmärgini



Joonistel on näha, et algoritm leiab teed isegi siis kui NPC ning sihtmärgi asukohad ära muuta

8.6.3 Hüppete simulatsioon ning vajadusel teise tee leidmine

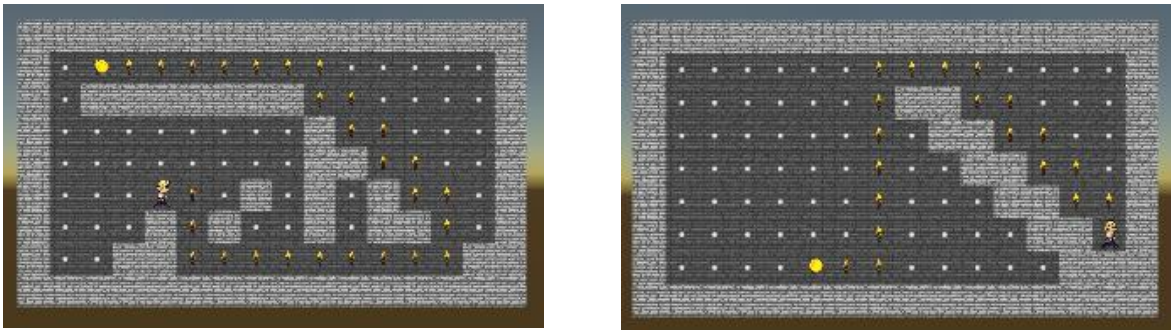
Tegelane ei saa hüpata üle kahe bloki ning kui sihtmärk on hüppamiskaugusest väljas, siis algoritm läheb mööda teist teed.



Joonistel on näha, et NPC läheb mööda pikemat teed (märgitud tõrvikutega) kuna ta ei suuda tema piirangute tõttu minna mööda lühimat teed (märgitud punase joonega)

8.6.4 Erinevad tasemed

Algoritm töötab siis kui muuta tasemete struktuuri.



Joonistel on näha, et kui muuta taset, siis tekkivad ikkagi õiged tipuobjektid ning A algoritm suutis leida teed sihtmärgini.*

9 Kokkuvõte

Eesmärgiks oli uurida 2D rajaleidmise algoritme ning realiseerida neist parim 2D platvormerilaadses keskkonnas. Antud projektis on valitud A* algoritm tema võime leida sihtmärk kasutades üht kõige lühemat teed.

Seejärel oli loodud Unity mootoril testimiskeskkond, kasutades uut *Tilemap* süsteemi ning asetada skriptidega keskkonnale läbimise tipud, mida algoritm kasutama hakkab. Pärast seda Algoritm leiab teed, arvestades 2D platformeri piiranguid ning sammhaaval viib *NPC'd* sihtmärgini. Testimise jaoks oli loodud mitmed erinevad tasemed, et veenduda, et algoritm vastab nõuetele.

Kokkuvõttes sai valmis testimiskeskkond, mis automaatselt skaneerib maatrikskujulise 2D platformeri taseme struktuuri ning viib *NPC'd* sihtmärgini, kasutades A* algoritmi.

Projekti raskusteks oli minu jaoks uue programeerimiskeele (C#) tundma õppimine ning Unity keskkonnaga tutvumine (Unity mootori kasutajaliidese õppmine, skriptide struktuur).

Lõputöö andis autorile ülevaadet Unity mootori struktuuri ning kasutajaliidese kohta. Lisaks sellele õppisin kasutama uut *Tilemap* elementi ning kuidas saab sellele pathfindingu algoritmi rakendada.

Projekti repositooriumi link asub Lisades

10 Kasutatud allikad

[1] 15 04. 2018. [Võrgumaterjal]. Available: https://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php.

[2] 14 05. 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine).

[3] 14 05. 2018. [Võrgumaterjal]. Available: https://www.nbmevents.uk/developawards2018/.

[4] 04 05. 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Taxicab_geometry.

[5] 06 05. 2018. [Võrgumaterjal]. Available: https://answers.unity.com/questions/238068/instantiate-prefab-and-initialize-its-variables.html.

[6] 13 05. 2018. [Võrgumaterjal]. Available: http://www.astrolog.org/labyrnth/algrithm.htm#perfect.

[7] 05 05. 2018. [Võrgumaterjal]. Available: https://stackoverflow.com/questions/24094093/how-to-print-2d-array-to-console-in-c-sharp.

[8] 14 05. 2018. [Võrgumaterjal]. Available: https://www.commander-keen.com/level-maps-6.php.

[9] 14 05. 2018. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/index.html.

[10] 14 05. 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm.

[11] 14 05. 2018. [Võrgumaterjal]. Available: https://www.redblobgames.com/pathfinding/a-star/introduction.html.

[12] 14 05. 2018. [Võrgumaterjal]. Available: https://www.raywenderlich.com/4946/introduction-to-a-pathfinding.

[13] 14 05. 2018. [Võrgumaterjal]. Available: https://gamedevelopment.tutsplus.com/tutorials/how-to-adapt-a-pathfinding-to-a-2d-grid-based-platformer-theory--cms-24662.

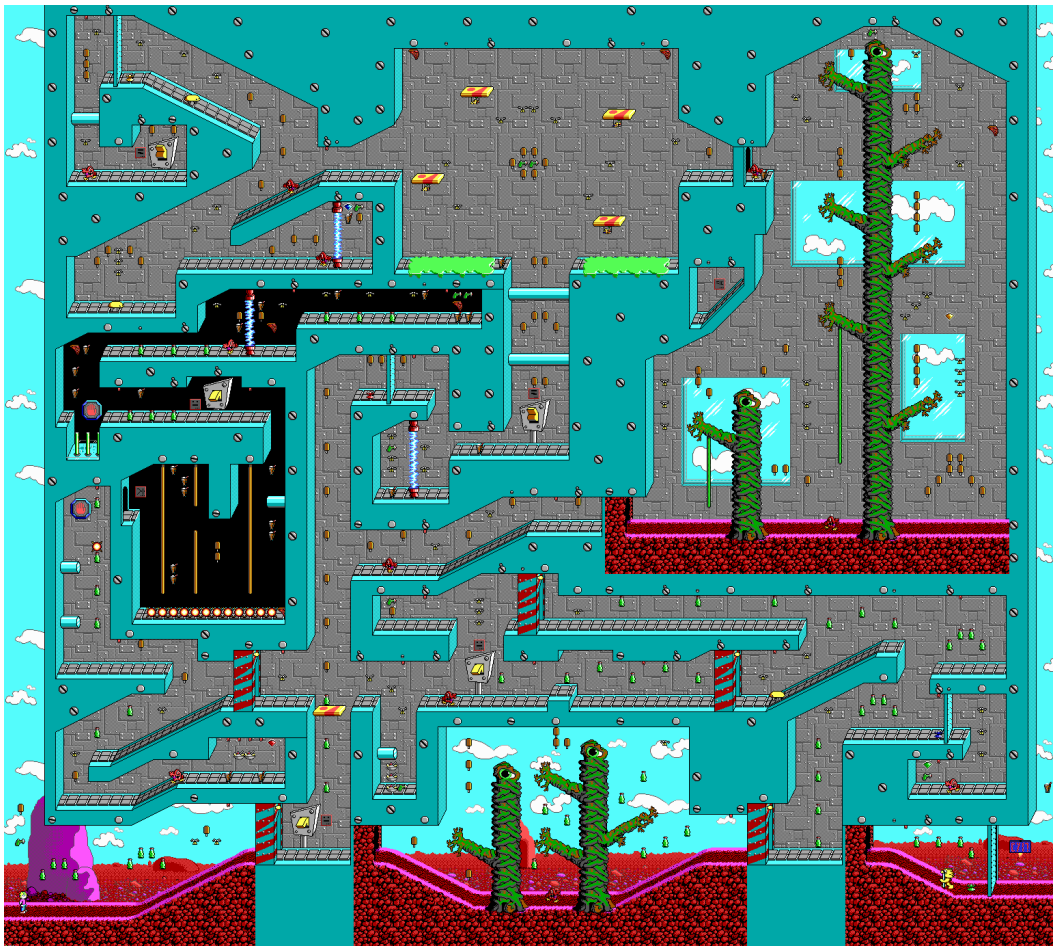
[14] 14 05. 2018. [Võrgumaterjal]. Available: https://unity3d.com/ru/learn/tutorials/topics/2d-game-creation/intro-2d-world-building-w-tilemap?playlist=17093.

[15] 14 05. 2018. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Platform_game.

[16] 16 05. 2018. [Võrgumaterjal]. Available: https://trends.google.com/trends/explore?date=all&q=how%20to%20make%20a%20game,unity%20tutorial,game%20engine.

[17] 19 05. 2018. [Võrgumaterjal]. Available: https://www.reddit.com/r/programming/comments/5u9yth/a_a_star_search_algorithm_computerphile/.

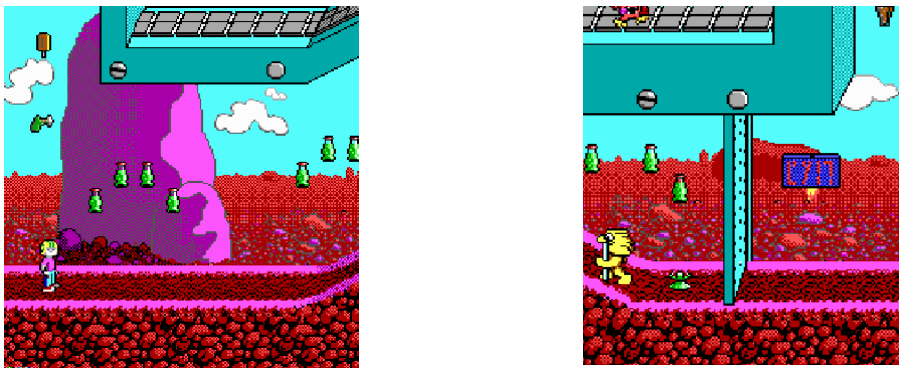
11 Lisa 1- (Klassikalise 2D platformeri näide)



"Commander Keen 6- Aliens Ate my Baby Sitter" tase nr. 10

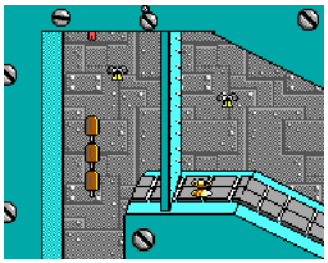
"Commander Keen 6" on hea näide platformeri mängust, kuna ta sisaldab kõiki (eelmainitud) platformerile iseloomulike elemente:

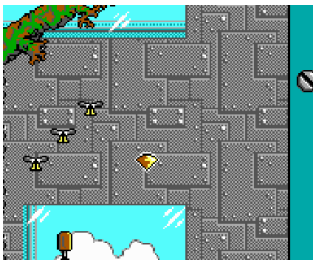
1) Peaülesandeks on liikumine taseme algusest taseme lõpuni



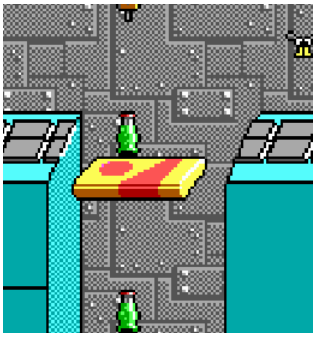
Taseme algus (koos peatgelasega) ning taseme lõpp

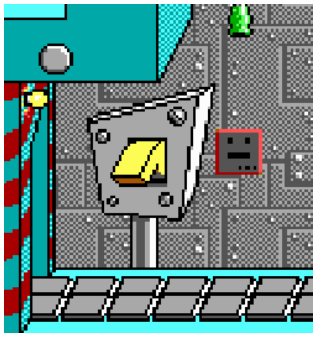
2) Lukustatud uksed ja nende võtmed





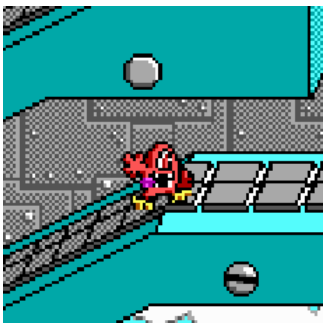
Kollane lukustatud uks ja kollane võti





Liikuv platvorm ja platvormi lüliti

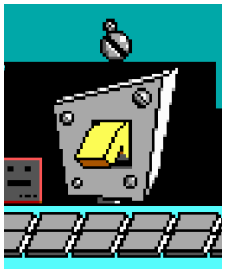
3) Vastased, kes soovivad tappa peategelast



Taseme vastased

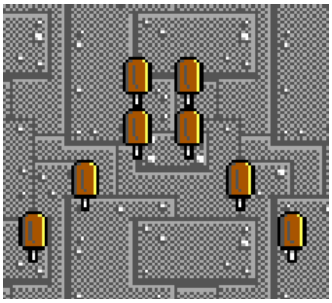
4) Lõksud ja nende lülitid





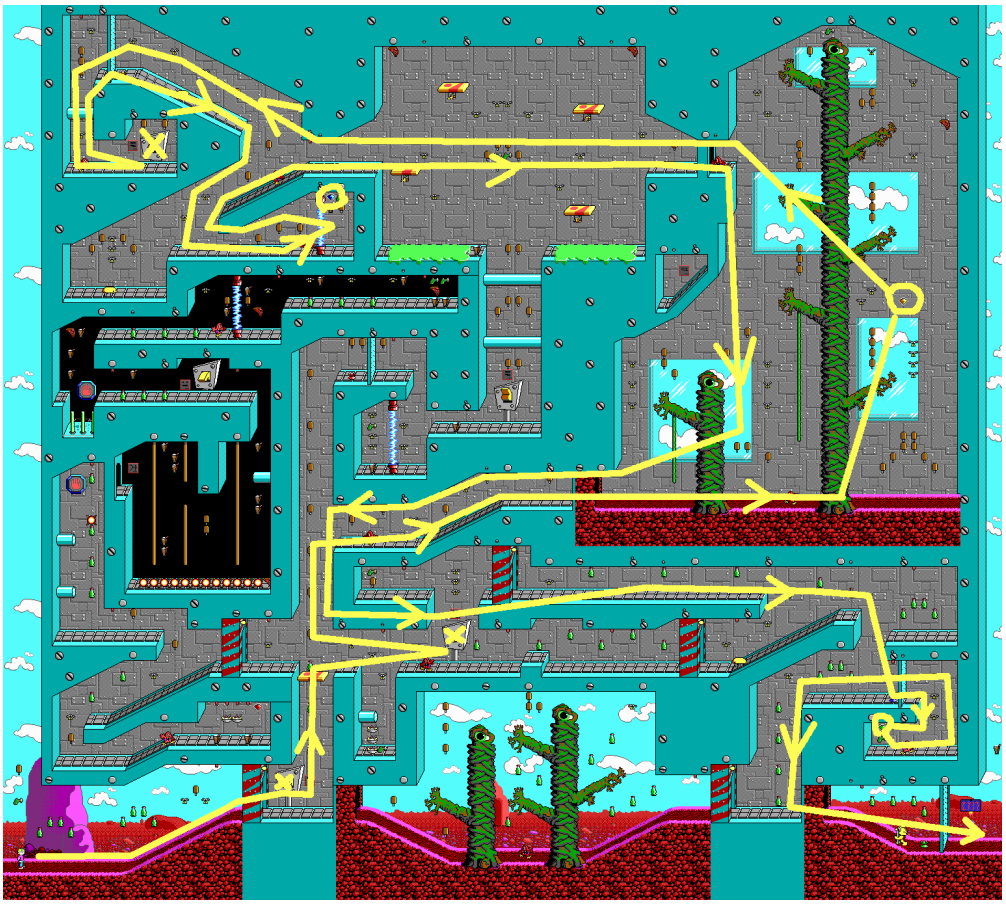
Lõksud, mida ei tohiks puutuda ja lüliti, mis lülitab lõksu välja

5) Boonused




Punktiboonus ja püstolikuulid

6) Labürinditaoline struktuur



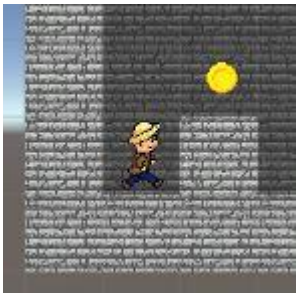
Teekond algusest lõpuni kasutades minimaalset hulka võtmeid ja ignoreerides boonusteruumi (kõige lühem teekond) [8]

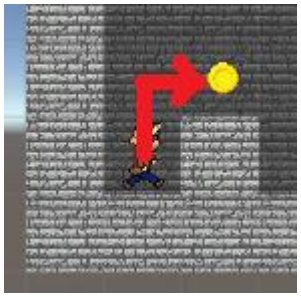
12 Lisa 2 – Link Github'i repositooriumile

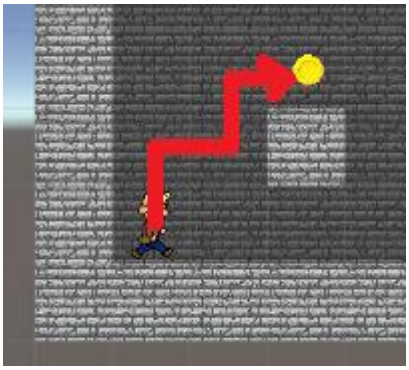
https://github.com/AlVeber/2D-platformeri-labimine-Astar-algoritmi-abil---Alain-Veeber.git

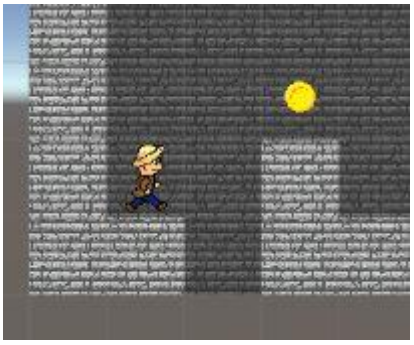
13 Lisa 3 – Hüpete näidised ja nende lahendused

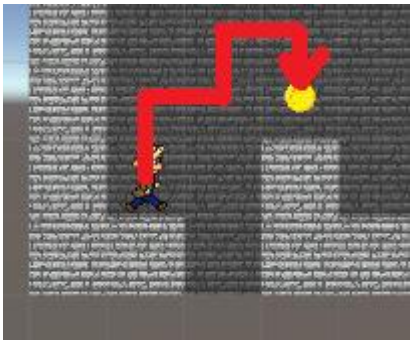
Siia on lisatud lubatud hüpete näidised. *NPC* ei tohi liikuda üle 2 bloki ülespoole ning tema Y-koordinaadi muutmisel saab ta üks kord liikuda paremale või vasakule

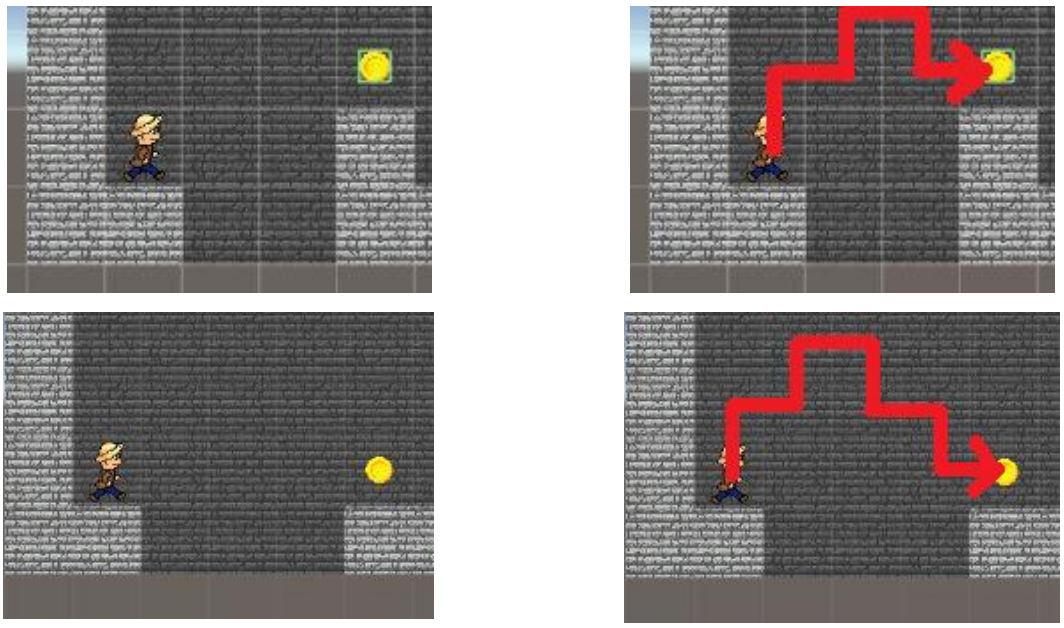












Lubatud hüpete näidised ja nende lahendused