Internet stvari u računalnim igrama

Sedlanić, Leon

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Leon Sedlanić

Internet stvari u računalnim igrama

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Leon Sedlanić

JMBAG: 0016136119

Studij: Informacijski sustavi

Internet stvari u računalnim igrama

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Boris Tomaš

Varaždin, rujan 2022.

Leon Sedlanić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Glavna tema ovog rada će biti povezanost računalnih igara i njihovih korisnika pomoću interneta stvari. Kao prvo moći će se saznati nešto više o početcima IoT - a i kako je integriran u naše živote te što je zapravo internet stvari, koje stvari spadaju pod tu mrežnu infrastrukturu a koje ne, tehnologije uz pomoć kojih se ostvaruje Internet stvari i slično. Druga stvar koja će biti obrađena jest zapravo glavni dio ove teme a to je Internet stvari u računalnim igrama, to jest početci, razvoj, tehnologije i primjeri.

Nakon toga dolazi praktični dio teme a to je izrada uređaja koji komunicira sa računalnom igrom preko interneta bilo da šalje podatke ili ih prima i obrađuje na neki način. Ja sam se odlučio za izradu uređaja koji će primati podatke od strane računalne igrice Dota 2 i prikazivati ih vizualno. Za taj dio će biti prikazana shema spajanja mikro kontrolera i potrebnog hardvera, nabrojana oprema, programski kod potreban za rad uređaja i primjeri funkcija te konkretnog rada kada komunicira sa računalnom igrom.

Sadržaj

1. Uvod

Internet stvari je poboljšao i olakšao veliki dio naših života pojedinačno, a i skupno sa povezanosti između raznih proizvođača i dobavljača, klijenata i poslužitelja, profesora i studenata, itd. Većina ljudi je upoznata sa stvarima kao što su Amazon - ova Alexa i Google Home, ali postoje puno drugih uređaja koji uglavnom ukomponiraju senzore te operiraju preko interneta stvari. Ono što se još nije potpuno komercijaliziralo jesu igre bazirane na internetu stvari. Unatoč tome, sve više rastući broj igrača video igara dovodi do sve više istraživanja u polju igara povezanih na Internet stvari.

Za ovu temu sam dobio motivaciju od toga što volim računalne igre od kada sam bio mali i smatram da su mi pomogle u razvoju tijekom mladosti (postoje istraživanja koja isto smatraju da računalne igre poboljšavaju kognitivne sposobnosti i potiču kreativno razmišljanje) tako da čim sam vidio naziv teme sam si nekako zacrtao u mozgu da ću to raditi iako sam gledao i neke druge teme ali sam se uvijek razmišljanjem vratio na ovu. Jednog dana želim raditi u razvoju igara i mislim da će mi ova tema donekle biti važna u mojem daljnjem educiranju, a možda i poslu nakon educiranja.

2. Metode i tehnike rada

Za izradu uređaja bilo je potrebno proučiti razne tehnologije koje su se koristile tijekom cijelog procesa kao što su web alat Tinkercad za izradu dizajna te dokumentacija Node.js te poneke github i arduino biblioteke. Također je bilo potrebno testirati mnogo puta funkcionalnosti uređaja i kako je uređaj baziran na računalnoj igri Dota 2, logično je da se je odigralo više nego nekoliko različitih rundi te igre.

3. Internet stvari

Za Internet stvari (eng. Internet of Things ili IoT) možemo reći da je dinamična globalna mrežna infrastruktura koja koristi samostalno konfiguriranje te ima osnovicu u standardnim i inter operabilnim komunikacijskim protokolima gdje fizičke i virtualne „stvari“ imaju fizičke atribute i virtualne osobnosti te koriste pametna sučelja. U mnogim slučajevima, te „stvari“ komuniciraju s podacima koja se odnose na korisnike i okoliš bilo korisnika ili stvari (Bahga, Madisetti, 2014).

Isti autori navode i karakteristike IoT uređaja:

- Dinamičnost i prilagodljivost – uređaji i sustavi mogu imati sposobnost dinamičkog prilagođavanja kontekstima i slučajevima i prema njima se odnositi na drugačije načine (npr. klima se može uključiti ili isključiti bazirano na temperaturi u sobi)
- Samokonfiguracija – veliki broj uređaja mogu raditi zajedno kako bi postigli neku funkcionalnost
- Interoperabilni komunikacijski protokoli – uređaji mogu imati mogučnost komuniciranja sa drugim uređajima pa i sa infrastrukturom
- Identitet – svaki uređaj ima svoj identifikator (IP adresa ili URL)

Još jedna definicija koju spominju Salazar i Silvestre (2017., str. 6) bi bila ta da je Internet stvari globalna tehnička arhitektura bazirana na internetu koja olakšava razmjenu dobara i servisa u globalnom lancu opskrbe.

Arhitektura svakog IoT sistema se može podijeliti na 4 razine: razina osjeta objekata koja prikuplja podatke o fizičkim objektima, razina razmjene podataka gdje se događa transparentni prijenos podataka kroz komunikacijske kanale, razina informacijske integracije unutar koje se procesiraju nesigurne informacije prikupljene iz mreže te se filtriraju neželjeni podaci i gdje se glavni podaci pretvaraju u oblike korisne krajnjim klijentima ili servisima za daljnju uporabu, razina servisa aplikacije u kojemu se stvara sadržaj namijenjen korisnicima. (Salazar i Silvestre, 2017., str. 13)

Samo neki od primjera interneta stvari bi bila pametna svjetla koja se pale i gase bazirano na vanjskoj osvijetljenosti, pametni televizori sa mogućnosti povezivanja na Internet, kamere koje reagiraju na provalnike i šalju SMS, detektiranje vodostaja pomoću senzora, sustavi za pametnu kupnju kao npr. keks pay, automati koji prate količinu preostalog inventara, sustavi za irigaciju koji prate razinu vlažnosti u zemlji, itd. Samo po ovim primjerima se vidi da IoT djeluje na cijeli svijet i povezuje ga na načine koji nisu bili ni zamislivi u nedavnoj prošlosti, a o budućnosti se može samo nagađati.

3.1. Evolucija

Početna točka interneta stvari a i samog interneta uvijek će biti ARPANET, ali to je daleko od samog početka interneta stvari kakvog mi znamo danas.

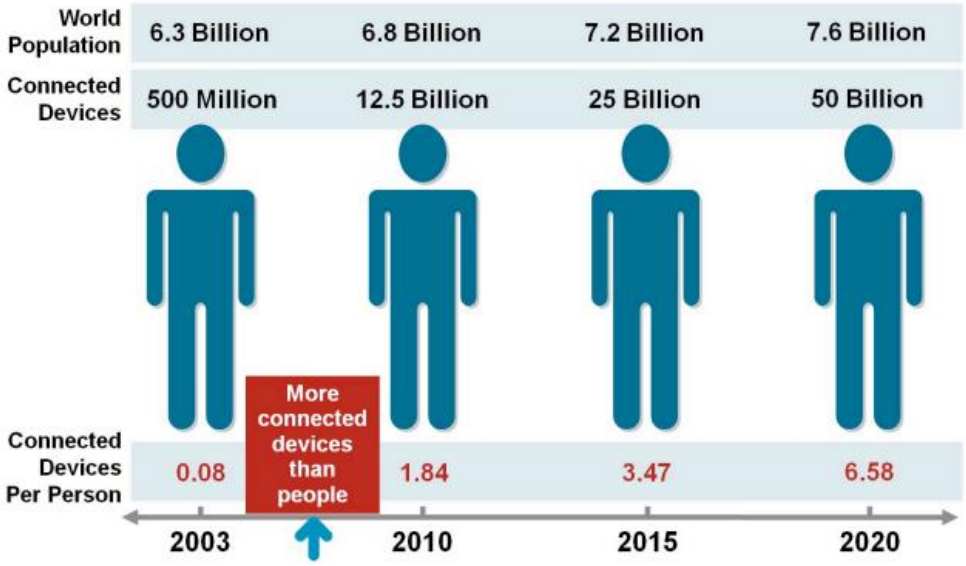
Internet je od svog početka do današnjeg dana u konstantnom razvoju, no kada pričamo o njegovoj funkciji, ona je ostala otprilike ista još otkada se je mreža zvala ARPANET i kada se je koristila primarno u akademske svrhe. Internet stvari, doduše, jest prva prava evolucija interneta u kojemu se je internet proširio na razna područja. (Evans, 2011., str. 5)

Godine 1982. student sveučilišta Carnegie Mellon zvan David Nichols želio je znati u bilo kojem trenutku da li ima u aparatu za koka kolu proizvoda i da li je on hladan. Zajedno sa svojih dvoje kolega Mikeom Kazarom i Ivorom Durhamom te sveučilišnim inženjerom Johnom Zsarnayom napisali su programski kod koji je upravo to provjeravao. Nakon toga bilo tko na sveučilišnom ARPANETU je mogao provjeriti status aparata za sokove. Još jedna prekretnica je naravno bio prijedlog za izradu okvira World Wide Web kojeg je predložio Tim Berners Lee godine 1989. što je bio početak interneta kakvog mi znamo, ali što se tiče interneta stvari, prva „stvar“ koja je ikad bila napravljena jest toster koji se mogao paliti i gasiti preko internetske veze kojeg je napravio 1990. godine John Romkey. (Techaheadcorp, 2020.)

Fraza „internet stvari“ je prvo bila spomenuta od strane Kevina Ashtona 1999. godine kada je radio u polju RFID (eng. Radio Frequency Identification – identificiranje radio frekvencija). (Salazar i Silvestre, 2017., str. 6)

Internet se je u to vrijeme tek počeo razvijati, a čovječanstvo još uvijek nije imalo sve na dohvatu ruke. Umjesto na mobitelima, kontakti su bili spremljeni u telefonske imenike, video isječci i filmovi su bili na kazetama, slike su postojale samo u fizičkom obliku osim u fotoaparatima i sličnim stvarima i nisu postojale web trgovine. Uzmemo li to sve u obzir, možemo reći da je pravi početak interneta stvari bio odmah nakon izuma pametnog telefona. Sve nabrojano se je tada moglo staviti u jedan uređaj koji je bio uvijek pri ruci sa mogućnošću povezivanja na Internet. U to vrijeme je naravno krenuo i rast društvenih mreža. Pričanje sa osobama na drugom kraju svijeta je do tada bilo gotovo nezamislivo bez satelitskih telefona, a društvene mreže kao npr. najpopularnija Facebook su omogućile gotovo trenutno čavrljanje sa drugim osobama preko mreže i dijeljenje fotografija, videa, slika, razmišljanja i sl.

Evans (2011., str. 3) ističe kako je Cisco grupa za internetske biznis solucije estimirala da je internet stvari „rođen“ negdje između 2008. i 2009. godine jer je u to vrijeme broj povezanih uređaja na Internet prešao broj cijele ljudske populacije tada na planeti. To možemo vidjeti bolje na sljedećoj slici koja to prikazuje grafički.



Slika 1. Evolucija IoT „(Prema: Evans, 2011.)“

IoT je poboljšao kvalitetu i performanse globalne logistike, pospješio ekonomiju, povećao kvalitetu života i unaprijedio rezultate još mnogo dijelova ljudske infrastrukture bilo na globalnoj razini, unutar države, grada, sela, zajednice, obitelji ili pojedinačno. Sve to je omogućeno zato što „stvari“ unutar interneta stvari kreiraju brže, efikasnije i kvalitetnije izlaze koji se mogu odmah uz pomoć internetske veze obraditi dalje ili na isti način obrađuju ulazne podatke koji bi bez interneta sporije se obrađivali.

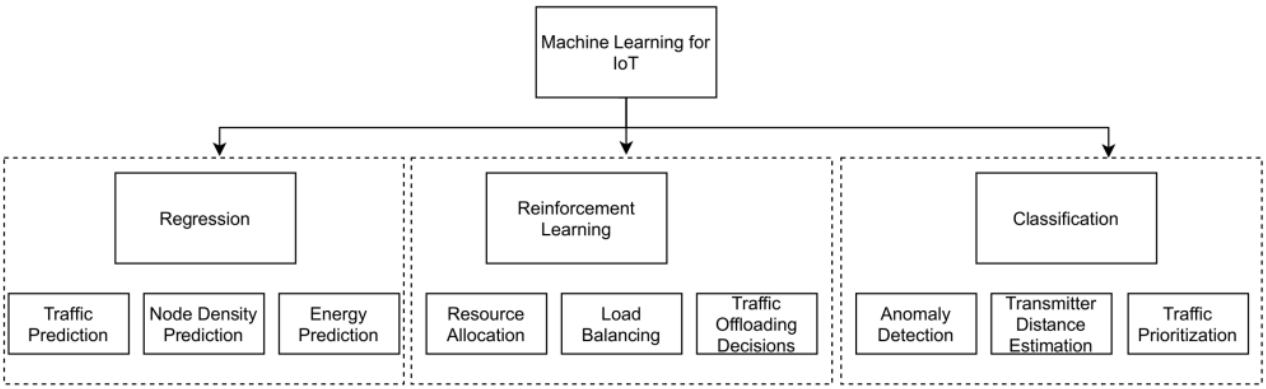
Napredak interneta stvari je omogućen uglavnom zbog tri specifične tehnologije koje su u prošlom desetljeću također vidjele veliki pomak. To su računarstvo u oblaku, veliki podaci i umjetna inteligencija. Zbog računarstva u oblaku moguće je spremati veliku količinu podataka u oblaku, pa i podatke generirane od IoT sustava. Tehnologije velikih podataka su omogućile lakše organiziranje i prepoznavanje podataka generiranih od strane IoT sustava, a umjetna inteligencija sa svojim uvidima u podatke je pomogla kompanijama koje koriste internet stvari da rastu i dalje. (Techaheadcorp, 2020.)

3.2. Budućnost

Govoreći o daljnjem razvoju interneta stvari, postoje tri glavna smjera u kojima se može razvijati. To su strojno učenje, 6G komunikacije te blockchain sigurnost. (Zubair Khan i sur., 2021., str. 15)

Strojno učenje

Kako se uređaji interneta stvari razvijaju tako se i prikuplja sve više i više različitih i kompleksnijih informacija koja trenutne tehnologije ne mogu kvalitetno i efikasno obraditi. Zato će u budućnosti biti potrebne tehnike strojnog učenja koje će analizirati te podatke i tako povećati pouzdanost IoT uređaja. Dijagram na slici 2 nam prikazuje tri različite tehnike strojnog učenja koje imaju slučajeve korištenja unutar interneta stvari. Regresijske tehnike pomažu na način da precizno predviđaju tok podataka koji se kreće po mreži da bi se prevenirali zastoji u protoku. Također mogu predvidjeti na taj način promet u nekom sustavu interneta vozila (internet stvari ali za vozila) te smanjiti broj zastoja. Slično tome se pokušava prevenirati nepotrebno gubljenje energije sa algoritmima koji predviđaju koliko energije sustav treba u kojem trenutku. Tehnike pojačanog učenja se koriste za bolju alokaciju resursa i balansiranje tereta u IoT sustavima. Algoritmi sadržani unutar ove tehnike pokušavaju prepoznati omjere rasterećenja zadataka, izbor snage prijenosa, i sl. Klasifikacijske tehnike kao što su k - najbliži susjed i stabla odlučivanja nalaze korisnost u rješavanju izazova poput detekcije anomalija za poboljšanu sigurnost. Pronalaze lažirane podatke i ometajuće signale te na taj način osiguravaju normalan rad IoT mreže. Uz to, mogu se koristiti i u aproksimaciji udaljenosti između odašiljača i prijemnika. (Zubair Khan i sur., 2021., str. 15)



Slika 2. Strojno učenje unutar IoT sustava „(Prema: Zubair Khan i sur., 2011.)“

6G komunikacije i Blockchain sigurnost

Trenutno najviše korištena tehnologija komunikacije u svijetu jest 4G, a nedavno se je krenula uvoditi i koristiti 5G tehnologija. 6G tehnologija komunikacije je trenutno u razvoju i biti će nasljednik takozvane 5G mreže. 6G tehnologija nadalje će pospješiti tok podataka, omjer isporuke paketa i latenciju. Također, koristit će teraherz komunikacije, rekonfigurabilne pametne površine (RIS) i masivno računanje. (Zubair Khan i sur., 2021., str. 16)

Postoji ogroman broj IoT uređaja i aplikacija trenutno aktivnih u svijetu koji su veoma bitni. To mogu biti aplikacije ili uređaji koji se koriste u praćenju zdravstvenih stanja, sigurnosti na cesti, logistici u prehrambenoj industriji, vojne aplikacije, itd. U slučaju da se neka od njih sruši zbog napada i prodora njihove zaštite, posljedice mogu biti katastrofalne. Zato je potrebna blockchain zaštita podataka zbog svoje dokazane efektivnosti. (Zubair Khan i sur., 2021., str. 16)

4. Internet stvari u računalnim igrama

Zadnjih dva desetljeća cijeli svijet se polako povezuje preko interneta na razne načine i sa raznim motivacijama za to povezivanje. No, računalne igre nisu bile glavni cilj i fokusiralo se je uglavnom na integriranju raznih aplikacija koje su odrađivale neke zadatke sa internetom radi općenitog poboljšanja kvalitete života ljudi. To se promijenilo kako je rasla popularnost računalnih igara, a pogotovo onih kojima je potrebna internetska veza za komuniciranje s ostalim igračima i poslužiteljima usluga. Najveći rast IoT - a u računalnim igrama se dogodio kada su nastali malo moderniji pametni telefoni. Prije toga su igre na mobitelu bile veoma jednostavne i većini ljudi nisu bile previše zanimljive niti intuitivne. Nakon stavljanja više senzora u mobitel kao npr. akcelerometra, žiroskopa, magnetometra, GPS, senzora svijetlosti i sl., igre na mobitelima su odjednom mogle komunicirati sa korisnikom na puno više načina i to je otvorilo jako puno mogućnosti.

Nakon toga, igre su se mogle sve više igrati na različitim platformama (Xbox, PC, PlayStation, Mobile) na način da su igrači profili dijeljeni na sve platforme preko internetske veze. To je u današnje vrijeme postala norma; ako internetska igra želi postati i ostati uspješna, mora omogućiti korisnicima da tu igru mogu igrati bilo kad i bilo gdje. Dobar primjer toga jest igra Genshin Impact koju je napravio studio HoYoverse. Genshin Impact omogućuje korisnicima igranje na bilo kojoj od idućih platformi: Android, iOS, PC, PS4, PS5. Također, igrači sa bilo koje platforme mogu komunicirati i zajedno igrati sa igračima sa bilo koje druge platforme.

IoT u industriji igara nije samo pomogao klijentima / korisnicima i poboljšao im iskustvo. Uz to, developeri i testeri su dobili razne načine komuniciranja jedni sa drugima, a i developeri sa korisnicima što je dovelo do unikatnih načina pretvaranja igara da pristaju željama igrača.

Nadalje, internet stvari je spojio igre sa platformama kao što su npr. Twitch i Youtube sa snimanjem u stvarnom vremenu. Twitch također pruža mogućnost interakcije gledatelja sa takozvanim streamerima (osobe koje kreiraju sadržaj na Twitchu) na razne načine. U nekim slučajevima gledatelji mogu čak i utjecati preko interneta što će se dogoditi u igri koju gledaju.

Ponajviše bitna prednost kod prelaska igara na internet stvari jest ta da kombiniraju zabavu i relaksaciju sa edukacijom, zdravljem pa i treniranjem. Dijelovi života ljudi koji su rijetko kada bili zanimljivi ili proaktivni sada mogu biti. Studenti mogu dobiti računalne igre tijekom svojih nastavnih programa kako bi na zabavan način bili educirani, ljudi kojima je potreban trening mogu nabaviti igru u kojoj da bi napredovali moraju trčati, skakati, plesati, udarati ili nešto slično. (Muskan, 2021.)

Što se tiče fizičkog dijela interneta stvari unutar industrije igara, uređaji su uvijek različiti ovisno o kojoj igri je riječ. Za neke igre postoji uređaj koji ima senzore i odjevne predmete radi kretanja kako u pravome životu tako i u igri. Neke igre pak imaju bežične kontrolere u obliku mača ili pištolja sa senzorima kako bi se što više uživjeli u osjećaj igranja. (Muskan, 2021.)

Jedan primjer IoT uređaja povezanog sa računalnom igrom bio bi Pip-Boy napravljen od strane studija Bethesda za svoju igru Fallout 4. Pip - Boy je uređaj koji se stavlja na podlakticu te služi za postavljanje pametnog telefona u svoje kućište na koji se onda preuzima aplikacija unutar koje su sadržane funkcije uređaja. Uz pomoć tih funkcija dok je Pip - Boy spojen sa igrom, osoba može kontrolirati određene događaje unutar igre kao što su: korištenje predmeta za liječenje, mijenjanje oružja, mijenjanje radio stanica i sl.

Slika 3. Pip Boy (cdn.mos, 2018.)

Činjenica je da postoji tržište za IoT uređaje povezane sa računalnim igrama koje se temelje na zabavljanju korisnika, no postoje li druge vrste igara koje koriste IoT a da nisu temeljene na zabavi? Da, postoje, ali su tek u istraživačkim fazama i nema puno primjera za njih. To su igre koje povezuju pravi život sa igrama i sljedećih nekoliko poglavlja obrađuju takve igre.

4.1. Prožimajuće igre

Jedna vrsta igri koje se trenutno istražuju u polju igara spojenih sa internetom stvari su takozvane prožimajuće igre (eng. Pervasive gaming). Prožimajuće igre proširuju iskustvo igranja u stvarni život – na gradske ulice, divljinu ili dnevne sobe. Senzori prikupljaju informacije o kontekstu igrača i to stvara iskustvo koje se mijenja ovisno što igrač radi, kako se osjeća i gdje se nalaze. Jedna od takvih igara je Human Pacman gdje korisnik mora trčati uokolo kako bi kontrolirao karaktera unutar igre. (Benford, Magerkurth i Ljungstrand, 2005.)

Osim toga, jako dobar primjer jest igra „Can You See Me Now?“. Ova igra je jedna od prvih koja je bila bazirana na lokaciji igrača. Kontekst igre je taj da su postojale dvije grupe igrača. Jedna grupa je bila na računalima povezana na Internet i kretala se po virtualnom prikazu pravoga grada uz pomoć strelica. Druga grupa su bili lovci u pravome životu u pravoj verziji grada koji je u igri i oni su morali loviti igrače koji su se kretali na računalima. Ako su im došli preblizu i „vidjeli“ ih, igrač na računalu koji je viđen bio je izbačen iz igre. Igrači su mogli koristiti tekstualno čavrljanje, a lovci otvoreni glasovni kanal kako bi komunicirali zajedno. Ovaj projekt je bio druga velika kolaboracija između Mixed Reality Lab iz sveučilišta u Nottinghamu i grupe Blast Theory. (Blast Theory, 2003.)

Slika 4. Can you see me now? (Blast Theory, 2003.)

Također postoje i računalno proširene stolne igre koje uzimaju suprotni pristup gradeći na već uspješnim društvenim igrama i obogaćuju ih sa beneficijama informatičkih tehnologija kako bi dohvatili hibridni način zabave. Na taj način moguće je dobiti najbolje od oba svijeta sa socijalizacijom i interakcijom među ljudima na jednoj strani, a računalnom logikom koja prati igru te postavlja atmosferu na drugoj strani. (Benford i sur., 2005.)

4.1.1. Prožimajuće igre i IoT

Razvoj računalnih igara je često ostavio korisnikove fizičke aktivnosti i socijalne interakcije u prošlosti. Igrači su limitirani na tipkovnicu, miš i kontrolere tijekom igranja. Kako bi se to preveniralo, postoji rastući trend da se dovedu fizička aktivnost i socijalne interakcije unutar igara uz to da se i dalje iskoriste računalni i grafički sustavi. U tu ulogu ulaze prožimajuće igre među kojima se nalaze igre unutar proširene stvarnosti (eng. AR ili Augmented Reality) te virtualne stvarnosti (eng. VR ili Virtual Reality). (Magerkurth, Mandryk, Cheok i Nilsen, 2005.)

Mogućnost proširivanja virtualnih okruženja u stvarnost je jako puno napredovala u zadnjih nekoliko godina. Proširena stvarnost spaja informacije iz stvarnog svijeta sa informacijama koje se ne nalaze u njemu pomoću standardnih ljudskih osjetila. Većina AR sustava preklapa svakidašnji prostor korisnika sa novim informacijama, no teže je napraviti istu stvar unutar virtualne stvarnosti uz pribavljanje informacija toj virtualnoj stvarnosti iz stvarnoga svijeta. Dva nedavna primjera potrošačkih uređaja koji istražuju takve mogučnosti su Sony - eva Playstation Camera i Microsoftov Kinect for Xbox One. Ove platforme unašaju podatke o slici igrača u pravome svijetu sa kamere ili reagiraju na te podatke – tj. koriste Motion Tracking. Stuart Cunningham je 2019. izradio maketu sučelja igre koja bi za svakog igrača prezentirala njegove osjećaje pomoću podataka iz senzora za otkucaje srca, raznih senzora za prepoznavanje izraza lica (osjećaja) te galvanometra za mjerenje električnog otpora na koži što je indikator stresa. Te informacije bi se prikazivale iznad glava karaktera unutar igre. (Cunningham i Henry, 2020.)

Slika 5. Maketa sučelja igre (Cunningham i Henry, 2020.)

Također i prije Motion Tracking uređaja postojali su kontroleri koji su prebacivali korisnikove pokrete ruku u igru. Ti kontroleri su bili najpopularniji za igre kao što su tenis, kuglanje, badminton, golf i slični sportovi sa jednostavnim pokretima ruke.

Još jedan uređaj koji se koristi za igre u virtualnoj stvarnosti jesu naočale za virtualnu stvarnost (eng. Head - Mounted Display ili HMD). Ovaj uređaj se stavlja na glavu i ima ekran ispred očiju korisnika. Većina njih se sastoji od ekrana i sustava za praćenje kretanja. Podosta ih također ima opciju spajanja na internet. Omogućuje veće uranjanje u svijet računalne igre na način da se okretanjem glave u stvarnosti okreće i glava karaktera unutar igre. No, računalne igre moraju biti posebno napravljene za takav uređaj. Najpoznatiji takav uređaj je Oculus Rift. (Henry i sur., 2017.)

Jedan primjer igre koja koristi te uređaje bi bio VRChat u kojoj igrači komuniciraju i igraju razne mini-igre sa igračima iz cijelog svijeta sa avatarima unutar igre. Ti avatari mogu imati razne značajke povezane sa stvarnim svijetom kao što su sinkronizacija usana, praćenje očiju i veliki spektar pokreta tijela.

Slika 6. VRChat avatari (Marcinkowski, 2022.)

4.2. Stručne igre

Stručne igre su igre kojima primarna svrha nije zabava. Koriste se u puno različitih sektora kao što su naprimjer edukacijski sektor, vojni sektor i zdravstveni sektor. Flatner i sur. (2021.) u svome istraživanju o mogućnostima korištenja stručnih igara uz pomoć interneta stvari navode kako je nekoliko studija dokazalo efektivnost stručnih igara. Također predstavljaju rezultate upitnika u kojemu je sudjelovalo 133 studenata između 21 i 26 godina. Rezultati nam govore kako je 35% sudionika imalo prijašnjih iskustava sa stručnim igrama, a 56 % ih je imalo iskustva sa IoT. Čak 83 % je odgovorilo da ili možda na pitanje da li bi koristili IoT uređaj za bolju kvalitetu nekog dijela života i 55 % od tog postotka je odgovorilo sa da na pitanje da li bi koristili stručne igre za istu svrhu.

Iako je kombiniranje stručnih igara i interneta stvari još u istraživačkim fazama života, možemo vidjeti da bi postojalo veliko tržište za njih na školskoj i fakultetskoj razini pa i unutar drugih službi koje drže razne obuke kao što su vojska, policija, medicina i slično.

Calvo - Morata i sur. (2021.) verificirali su pozitivne učinke stručne igre koja je bila napravljena da bi se podigla svijest u vezi nasilja u školama. Nakon evaluacije uz pomoć upitnika i analiza u nekoliko škola, podigla se svijest na način da su djeca razmišljala o svojim djelima za koje su smatrali da su nebitni. Rezultati upitnika su pokazali da je igra imala pozitivan učinak na 80 % igrača.

Također, ovu efektivnost podupiru Wouters i sur. (2013.) koji su iskoristili pretraživanje literature kako bi istražili jesu li stručne igre više efektivne za učenje. Unutar istraživanja, pokazalo se je da su stručne igre daleko bolje od tradicionalnih instrukcijskih metoda.

Simoes i sur. (2018.) razvili su stručnu VR igru kako bi pomogli osobama na spektru autizma. Unutar igre simulirali su se svakidašnji događaji kao planiranje rute i interakcija s drugima na toj ruti. Anksioznost se je mjerila uz pomoć elektrodermalne aktivnosti.

4.2.1. Stručne igre i IoT

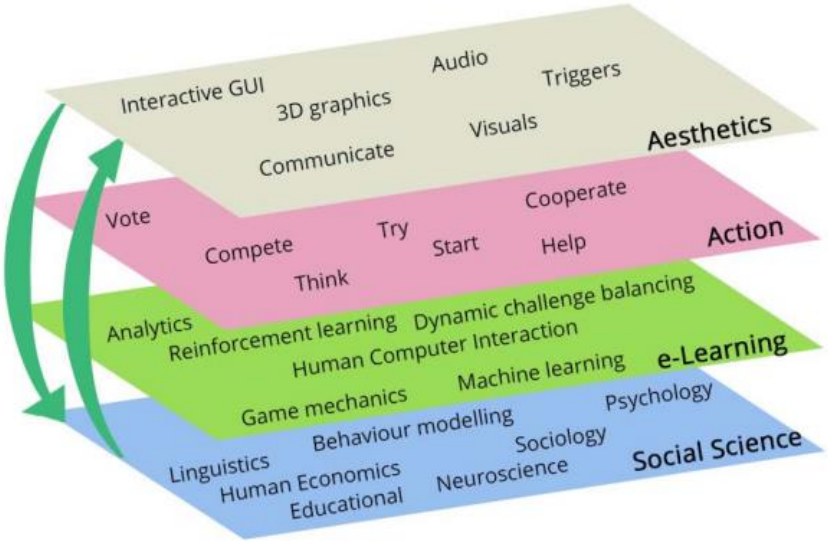
Evidentno je da računalne igre mogu biti puno više od softvera za zabavu. Maines i sur. (2015.) predstavili su pokušaj preklapanja zahtjeva sigurnosti sustava u ranim fazama dizajna uz pomoć Unity 3D igraćeg motora ili eng. game engine. Da uspostave IoT unutar istraživanja, mogli bi pristupiti fizičkim i virtualnim sustavima te prikazati ranjivosti unutar trenutačnih standarda.

Najveći tehnički izazov kombiniranja stručnih igara i interneta stvari jest heterogenost. Zbog razne kompeticije generira se segregacija i razvoj igara postaje glomazan. (Cunningham i Henry, 2020.)

4.2.1.1. Projekt InLife

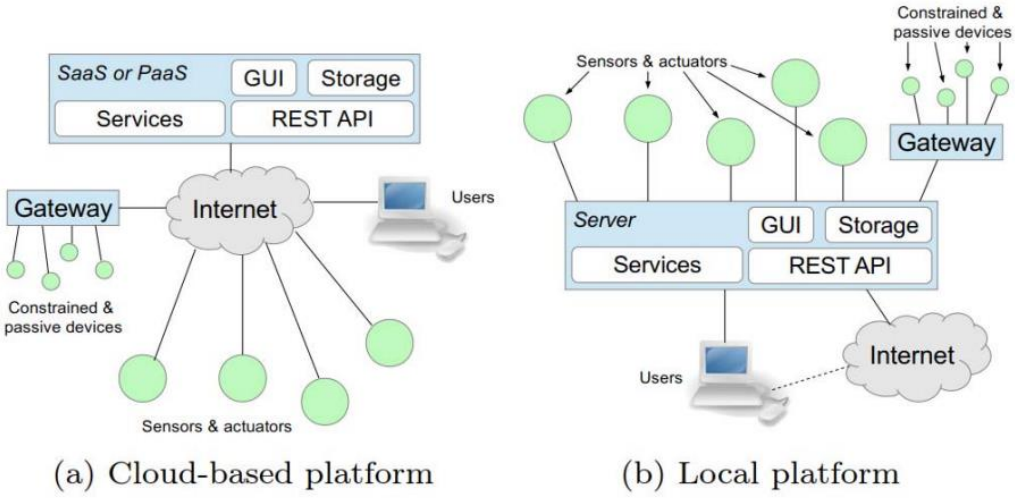
Tema ovog poglavlja je istraživanje koje se je provelo veoma nedavno. Unutar zadnjih nekoliko godina, razne kompanije su započele svoje projekte kombiniranja igara i interneta stvari. Jedan takav primjer je europski projekt InLife koji je napravljen kako bi istražio nove scenarije učenja unutar stručnih igara. Financiran je od strane EU i fokusira se na stručnim igrama i internetu stvari. Ovaj projekt daje uvid u kreiranje IoT stručnih igara.

Kada govorimo o okviru InLife - a, fokus je na olakšavanju integracije informacija iz stvarnog svijeta u svijet igara i potvrditi kako ovaj pristup pomaže u stvaranju impresivnih, prožimajućih stručnih igara koje mogu imati značajan utjecaj na njihovu namjeravanu obrazovnu učinkovitost. Ovo će biti ispitano na temelju implementacije „gamifikacije“ platforma i prateći razvoj stručnih igara, u kojem korisnici mogu napredovati dovršavajući određene zadatke u stvarnom životu, kao što je gašenje svjetla pri izlasku iz sobe ili surađivati s drugim osobama u jednostavnim radnjama. InLife želi ostvariti okvir za igranje koji bi također služio za edukacijske i socijalne svrhe povezivajući napredak unutar igre sa dijelima i odlukama u pravome životu koje bi bile detektirane od strane IoT infrastrukture uz pomoć pametnih senzora. (Kosmides i sur., 2018.)



Slika 7. Inlife u ljusci oraha „(Prema: Kosmides i sur., 2018.)“

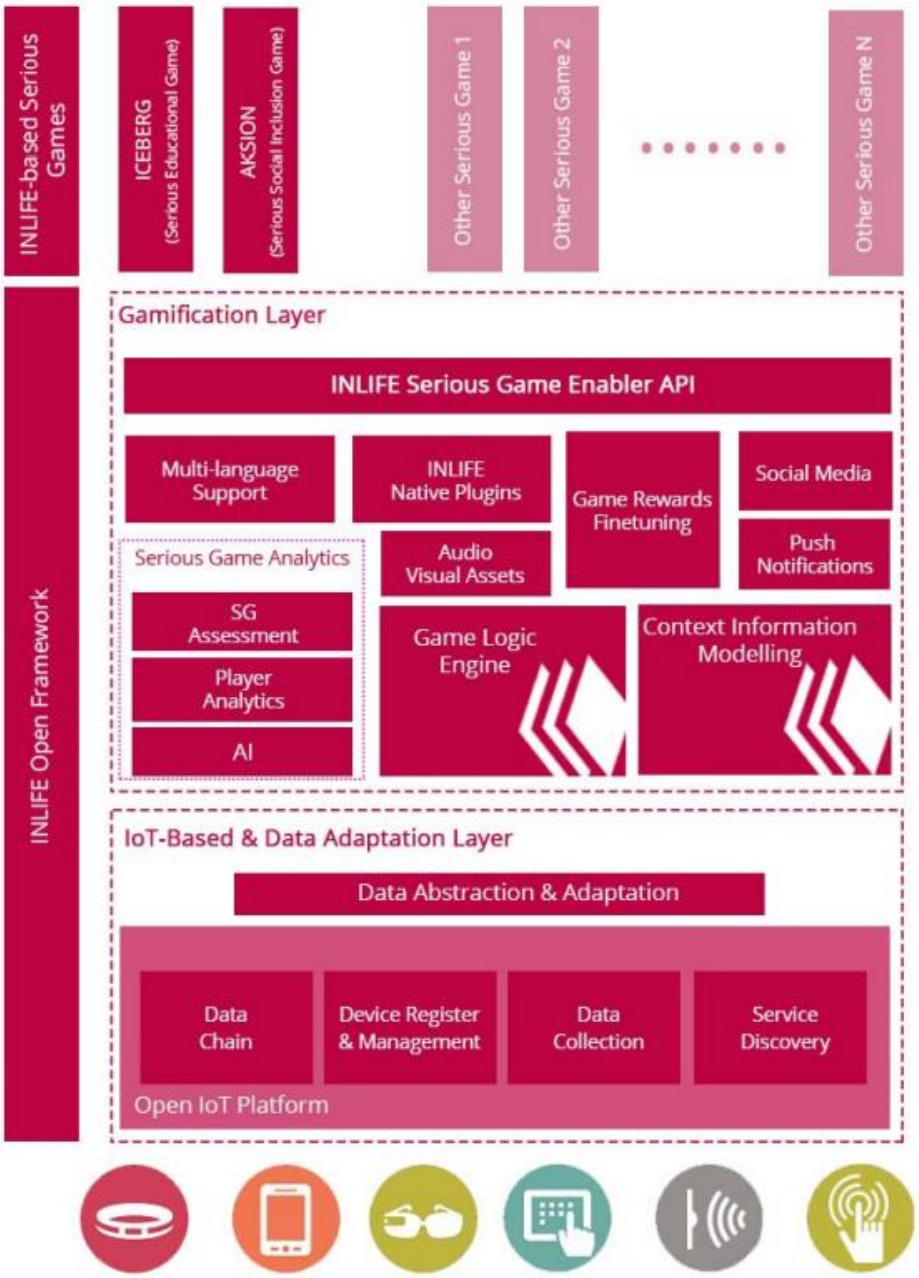
Jedna od važnih stvari u vezi gradnje IoT arhitekture za izradu igara koja se spominje u projektu InLife jest izbor između platforme koja je lokalna ili bazirana na oblaku. To je prikazano u sljedećoj slici:



Slika 6. Izbor arhitekture IoT „(Prema: Kosmides i sur., 2018.)“

Arhitektura InLife projekta je izrađena u dva dijela. Jedan dio je nazvan IoT - based Data Adaptation Layer koji obrađuje i dopušta komunikaciju između pametnih uređaja te preuzima agregaciju i adaptaciju podataka, a drugi dio je nazvan Gamification Layer u kojemu se koordiniraju servisi i kontrola „gamifikacije“. (Kosmides i sur., 2018.)

Ova arhitektura je također prikazana na slici koja slijedi.



Slika 8. InLife arhitektura „(Prema: Kosmides i sur., 2018.)“

Sve u svemu, InLife nam daje veoma dobar uvid kako bi trebala stručna igra bazirana na internetu stvari funkcionirati. Puno je različitih faktora o kojima treba voditi brigu i definitivno nije isto kao izrada običnih računalnih igara bile one stručnog tipa ili ne. U ovome projektu također je prezentiran njihov prvi slučaj korištenja u obliku stručne igre nazvane ICEBERG u kojoj igrači ako žele napredovati moraju biti svjesni svog okolišat e raditi ili ne raditi određene stvari u pravome svijetu kao što su npr. gašenje svjetla kada napuste sobu, gašenje računala kada nije potrebno za rad, korištenje stepenica umjesto dizala, zatvaranje prozora kada je klima upaljena i sl.

4.3. Aplikacijska programska sučelja računalnih igara

Jedan način na koji se može ostvariti IoT uređaj povezan sa računalnom igrom je uz pomoć aplikacijskog programskog sučelja za tu igru.

Aplikacijsko programsko sučelje ili eng. Application programming interface (API) je veza između aplikacija. Koristi se u integriranju jedne aplikacije unutar druge uz pomoć predodređenih pravila.

Način funkcioniranja svakog API-a je sljedeći: Kao prvo klijentska aplikacija započinje poziv API - a kako bi dobila neke informacije, tj. šalje zahtjev. Nakon što dobije važeći zahtjev, API radi poziv prema određenom serveru ili programu koji obrađuje taj zahtjev. Server ili program u sljedećem koraku šalje odgovor API - u sa traženom informacijom ili informacijama te na kraju API prebacuje te informacije nazad aplikaciji koja ih je isprva tražila. (IBM, 2020.)

Nemaju sve računalne igre svoj API niti su svi API - i koji postoje dobro dokumentirani, a ako API nije dovoljno dobro dokumentiran, teško da će itko osim osobe ili tima koji je izradio API shvatiti kako se ono koristi za integraciju u nekoj drugoj aplikaciji. Često kompanije ne vide svrhu u izradi sučelja, no može biti i da se boje zlouporabe tih sučelja što se i zna događati. Jedan primjer zlouporabe API - a igre jest PUBG (PlayerUnknown's Battlegrounds). Prije nekoliko godina, točnije 2018. godine, studio koji je napravio igru izbacio je u javnost i API za tu igru koji je mogao izvlačiti podatke u stvarnom vremenu o igračima. Inače unutar partije te igre, nijednom igraču nisu poznate informacije o drugim igračima dok se direktno ne nađu unutar runde. Pomoću API - a, pojedinci su mogli izraditi aplikacije koje su izvlačile informacije o drugim igračima u stvarnom vremenu i pomoću toga zapravo varati, što je dovelo do toga da je studio morao promijeniti API i onemogućiti podatke u stvarnome vremenu. To je bila velika šteta jer bi sigurno bilo u budućnosti mnogo zanimljivih projekata koji bi se mogli napraviti.

Neke od igara koje imaju veoma dobre API dokumentacije su:

- League of Legends
- Dota 2
- PUBG
- Fortnite
- Hearthstone
- Guild Wars 2
- Escape from Tarkov

4.4. Dota

Dota je započela kao nadogradnja računalne igre Warcraft 3 od strane njenih obožavatelja. Izrada dote je pokrenula novu eru računalnih igara tako što je nadodana nova vrsta igre u koju spada sama Dota – Multiplayer Online Battle Arena (MOBA). No, igra na kojoj će biti bazirana izrada uređaja unutar ovog rada je njezin nasljednik Dota 2. Dota 2 je nastala nakon što su kompanija Valve i dizajneri njihove igre Team Fortress 2 oko 2009. godine pokazali interes u izradu drugog dijela jedne od najpopularnijih igara u to vrijeme. U slično vrijeme se je krenula raditi i igra League of Legends koja je isto bila bazirana na prvoj Doti, i vrlo brzo je sa svojim posebnim stilom i načinom igranja pokupila više obožavatelja nego Dota. Nakon League of Legends i Dote 2, na tržištu su se krenule pojavljivati razne kopije i pokušaji otimanja baze igrača od te dvije igre, ali nijedna nije došla blizu.

Dota 2, slično kao i Dota, League of Legends i sve ostale igre koje spadaju u njihovu kategoriju igraju se u rundama po deset ljudi gdje su ti ljudi posloženi u timove pet protiv pet. Igra se iz ptičje perspektive sa kretanjem uz pomoć miša i pokojom kontrolom na tipkovnici. Na početku runde svako bira svojeg karaktera, a svaki karakter ima svoje prednosti i mane te moći koje mu pomažu u ostvarivanju svojih ciljeva unutar igre. Svaki član tima ovisno o karakteru i dogovoru između igrača ima svoje dužnosti prema ostatku tima te se na taj način ostvaruje timska igra kako bi pobijedili suparnički tim, tj. kako bi ostvarili glavni cilj, a to je uništenje suparničke baze. Također im u tome pomaže razna oprema koja pojačava karaktera, taktike i strategijske lokacije koje se nalaze po bojištu. Bojište se sastoji od tri linije po kojoj idu obični vojnici svakog tima unutar nekih intervala te džungle između njih unutar koje se mogu ubijati čudovišta za razne bonuse. Po svakoj liniji raspoređene su tri kule svakog tima te se one moraju uništiti prije nego se može nastaviti dalje na glavnu građevinu unutar suparničke baze. Sve u svemu, Dota je igra koja zahtjeva inteligenciju, taktiku i reflekse kako bi osoba nadjačala ljude iz suparničkog tima. Izgled bojišta se može vidjeti na sljedećoj slici:

Slika 9. Dota 2 karta igre

4.5. Dota 2 GSI

Za povezivanje uređaja sa računalnom igrom koristi se integracija sa stanjem igre ili eng. Gamestate Integration (GSI). Svakom Dota 2 klijentu je omogućeno uspostavljanje veze između klijenta i servera za GSI koji sluša i očitava događaje. No, moguće je samo očitavanje lokalnih događaja, ne i događaja drugih igrača unutar igre kako bi se preveniralo varanje. Za implementaciju korištena je biblioteka za Node.js koja pruža sučelje za Dota 2 GSI. Izradio ju je github korisnik xzion te se može naći na sljedećem linku: https://github.com/xzion/dota2-gsi.

Jedina druga igra za koju postoje biblioteke koje omogućuju integraciju sa stanjem igre jest Counter Strike Global Offensive, koju je također izradio Valve.

5. Izrada uređaja

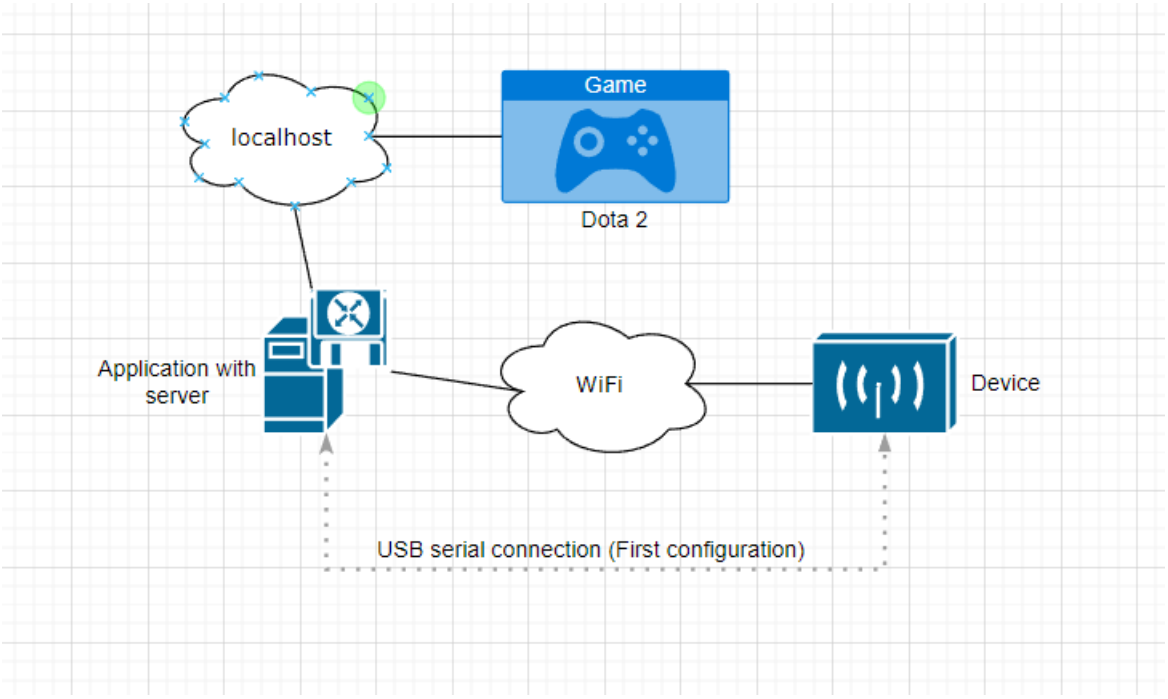
Za izradu uređaja i aplikacije na računalu bile su korištene sljedeće tehnologije:

- Node.js
- Arduino IDE
    - Adafruit_Neopixel.h za korištenje RGB LED trake
    - EEPROM.h za učitavanje podataka o WiFi mreži
    - ESP8266WiFi.h za spajanje na WiFi mrežu
- Visual Studio Community 2022
- Dodatne biblioteke za Node.js
    - @node-steam/vdf: verzija 2.2.0
    - dota2-gsi: verzija 1.0.2
    - fs-extra: verzija 10.1.0
    - serialport: verzija 10.4.0
    - winreg: verzija 1.2.4
    - pkg: verzija 5.8.0

5.1. Arhitektura, dizajn uređaja i komponente

Arhitektura sustava se sastoji od računalne aplikacije koja je u isto vrijeme poslužitelj spojen na localhost uz pomoć Dota 2 GSI biblioteke i klijent koji komunicira sa mikro kontrolerom ako je spojen na WiFi. Poslužitelj unutar aplikacije čeka odgovor računalne igre ako je otvorena na računalu. Igra cijelo vrijeme dok je uključena šalje podatke tom poslužitelju. Pločica sa mikro kontrolerom ima ulogu poslužitelja koja započinje kada se uspješno spoji na WiFi mrežu.

Ako pločica nije konfigurirana za trenutnu WiFi mrežu, potrebno ju je samo jednom spojiti sa USB kabelom radi konfiguracije i kako bi se pri daljnjem pokretanju povezivala sa tom WiFi mrežom.

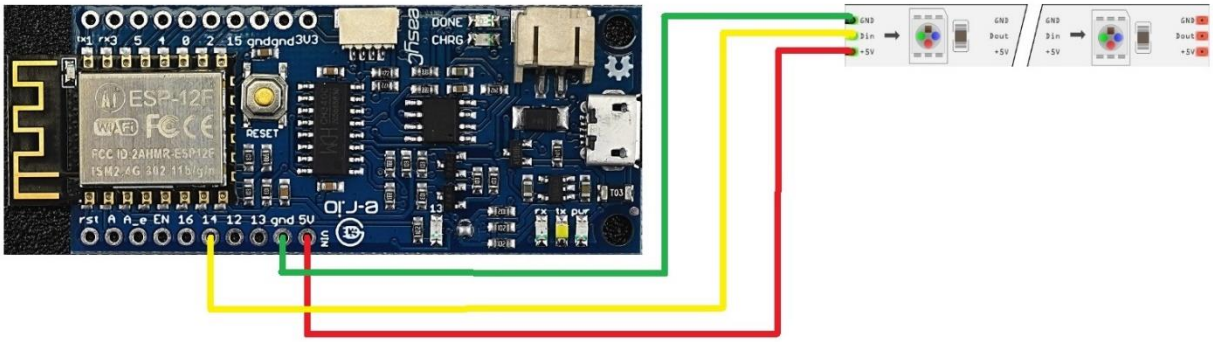


Slika 10. Arhitektura sustava

Za izradu uređaja korištene su sljedeće komponente:

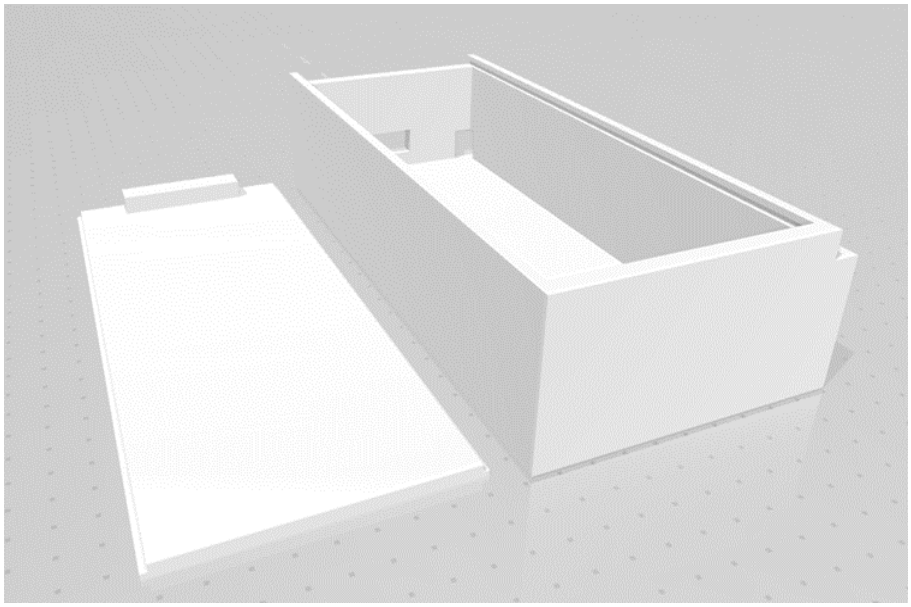
- eksperimentalna pločica (breadboard)
- Croduino NOVA2 - mikrokontrolerska pločica
    - ESP8266 u pakiranju ESP-12
    - 80MHz, 1MB flash, 82kB RAM
    - 9 GPIO pinova od kojih svi podržavaju PWM
    - 1 analogni ulaz(ADC), max. napon 1V i 5V
    - I2C, SPI, serijska komunikacija
    - CH340 USB na UART bridge s micro USB konektorom
    - 3.3V regulator napona
    - LED diode za: napajanje, tx, rx, GPIO13
    - hardwareski automatski reset prilikom uploada
    - dva pushbuttona: reset i GPIO0
    - FCC i CE certifikati za ESP-12
- RGB adresabilna LED traka

Shema spajanja komponenata prikazana je na sljedećoj slici – GND na GND (zeleno), 5V na 5V (crveno) i željeni pin na Din (žuto).

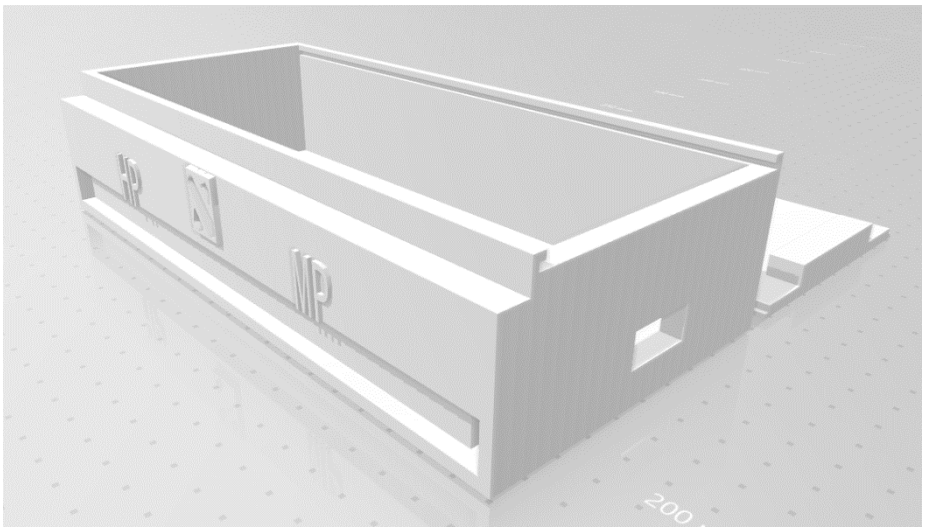


Slika 11. Shema spajanja komponenata

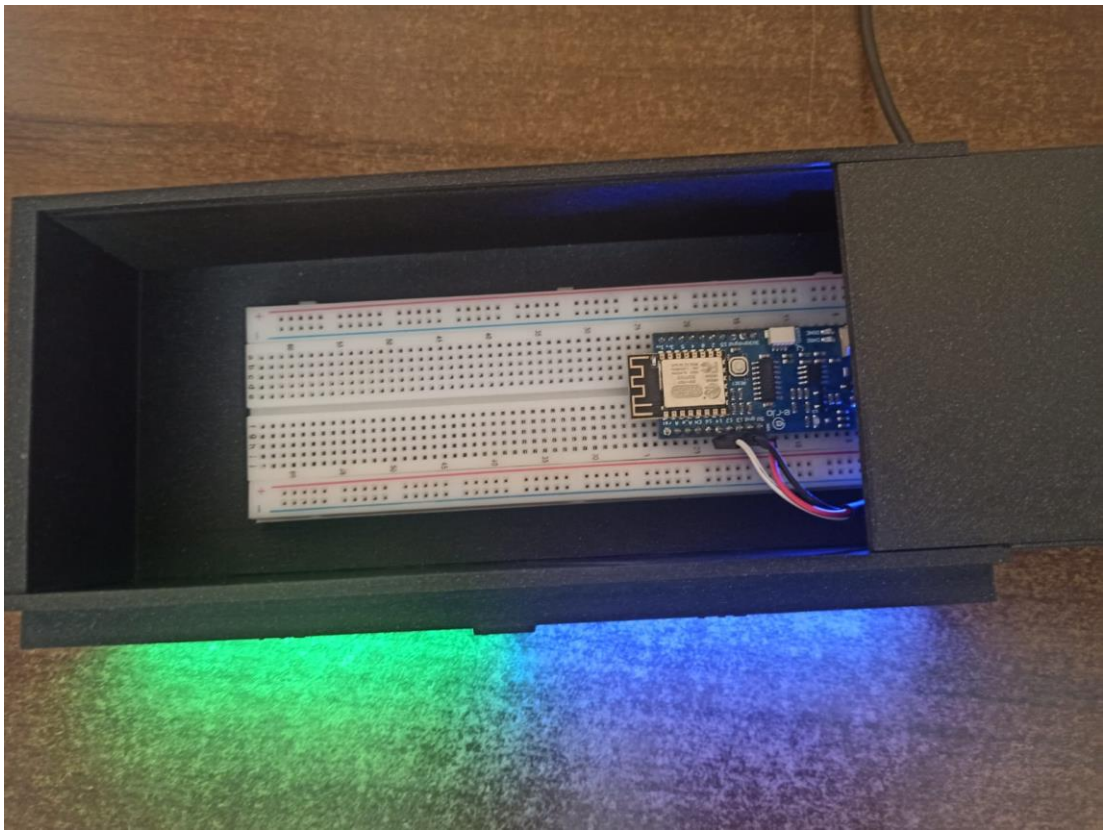
Dizajn uređaja je izrađen pomoću web alata Tinkercad, a sam uređaj je isprintan uz pomoć 3D printera. Dizajniran je tako da mikro kontroler na eksperimentalnoj pločici može stati unutar kutije te da se LED traka može provući kroz otvor u jednome kutu. Također je napravljena pravokutna rupa sa jedne strane kutije gdje je predviđeno mjesto za USB kabel.



Slika 12. Dizajn uređaja gledano sa stražnje strane

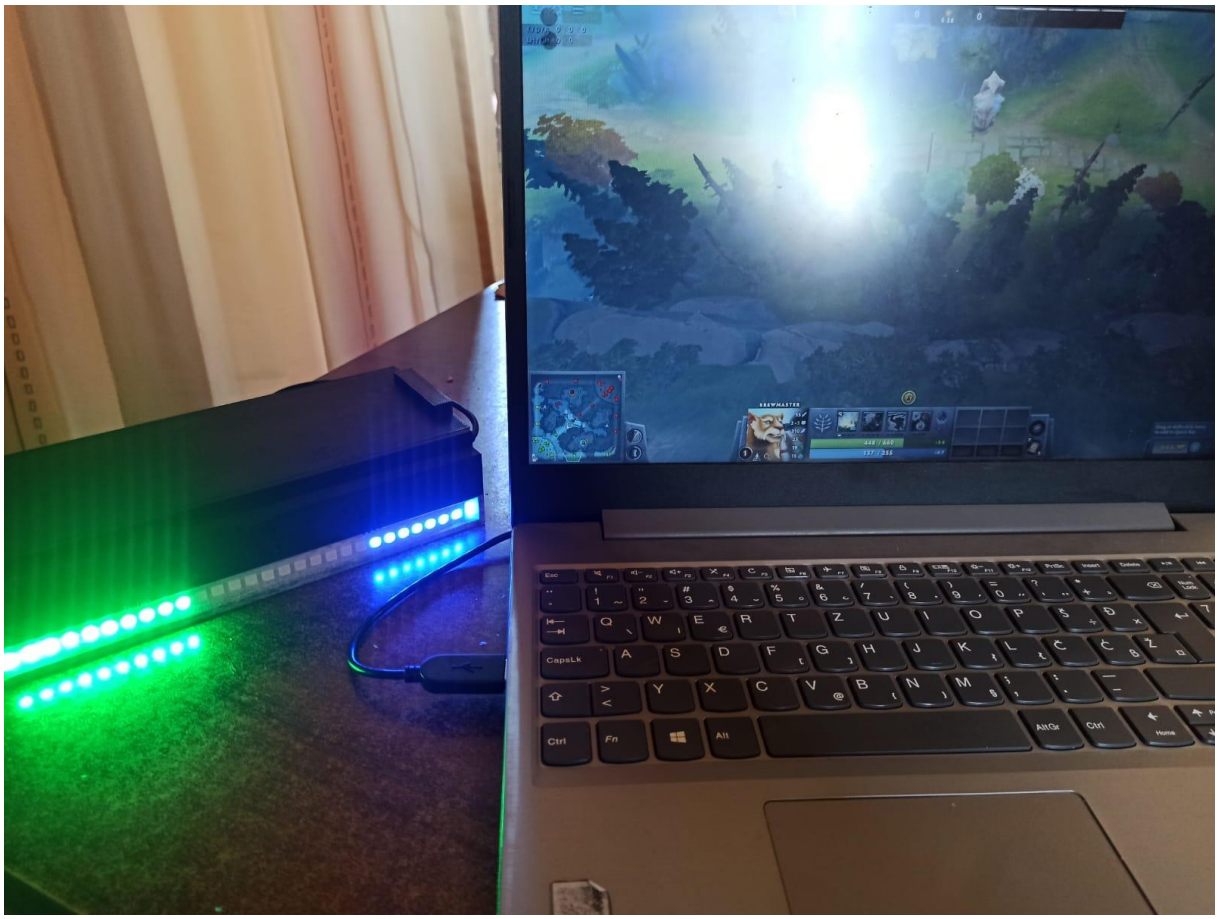


Slika 13. Dizajn uređaja gledano sa prednje strane



Slika 14. Izgled uređaja (unutarnji dio)

Slika 15. Uređaj povezan sa igrom



Slika 16. Prikaz promjena unutar igre

Na prijašnjim slikama napajanje uređaja je omogućeno uz pomoć USB punjača za mobitel.

5.2. Izrada Node.js aplikacije

Za izradu desktop aplikacije je korišten Node.js te Visual Studio kao IDE.

Node.js je open-source JavaScript runtime okruženje na više platformi. Koristi V8 JavaScript što je srž internetskog preglednika Google Chrome bez korištenja samog preglednika. Izvodi se u jednom procesu bez kreiranja nove dretve za svaki zahtjev. Kada je potrebno napraviti ulaz/izlaz operaciju, umjesto da blokira dretvu i zauzme procesor sa čekanjima, Node.js će nastaviti sa operacijama kada dobije nazad odgovor. (nodejs.dev, 2022.)

Uz neke osnovne biblioteke sadržane unutar Node - a, korišteno je i par vanjskih biblioteka koje se koriste u sljedećih nekoliko poglavlja.

5.2.1. Konekcija sa mikro kontrolerom

Kako bi ostatak programa funkcionirao, moraju se deklarirati sljedeće globalne varijable.

```
var d2gsi = require('dota2-gsi');
var server = new d2gsi();
const net = require('net');
const fs1 = require('fs');
const port = 80;
var host;
var isConnected = false;
var rawRequest;
var Pr = require('promise');
var readFile = Pr.denodeify(require('fs').readFile);
const { SerialPort, ReadyParser } = require('serialport');
const { ReadlineParser } = require('@serialport/parser-readline');
var arduino_com_port = 'comX';
const VDF = require('@node-steam/vdf');
const Registry = require('winreg');
const fs = require('fs-extra');
const path = require('path');
const os = require('os');
const REG = 'SteamPath'
const WINSTEAMLIB = path.join('steamapps', 'libraryfolders.vdf');
const GSIFILENAME = 'gamestate_integration_dota2-gsi.cfg'
const HOMEDIR = (process.env.SNAP) ? path.join('/home', process.env.USER)
: os.homedir()
const readline = require('readline').createInterface({
     input: process.stdin,
     output: process.stdout
});
```

Prva stvar kod izrade aplikacije jest omogućiti joj povezivanje sa pločicom. Na početku pokretanja aplikacije, ona se pokušava spojiti na prijašnju konfiguraciju koja je dobila sa pločice. Konfiguracija je spremljena u obliku .json datoteke koja nastaje nakon konfiguriranja WiFi - a sa pločicom. Ako još ne postoji ta datoteka, aplikacija nastavlja dalje.

```
const socket = new net.Socket();
fs1.exists('./saved_ip.json', function (exists) {
     if (exists) {
          readJsonFile('./saved_ip.json')
               .then((json) => {
                    host = json;
                    console.log("Trying to connect to board with previous
configuration...");
                    socket.connect(port, host);
               })
               .catch();
     }
     else {
```

```
            var again = false;
            console.log("Previous configuration not found, starting search
for arduino on serial ports...")
            findArduino();
            setTimeout(function () { IfFound(again); }, 7000);
      }
});
```

Postojanje datoteke se provjerava uz pomoć funkcije exists iz biblioteke fs-extra. Ako postoji, čita se njezin sadržaj uz pomoć funkcije readJsonFile i ako je datoteka uspješno pročitana, aplikacija će se pokušati spojiti sa IP adresom koja je sadržana unutar saved_ip.json. Za funkciju su također potrebne biblioteke promise i fs.

```
function readJsonFile(jsonName) {
      return readFile(jsonName, 'utf8').then((response) => {
            return JSON.parse(response);
      });
}
```

Ako datoteka ipak ne postoji, aplikacija pokreće funkciju findArduino() koja pokušava pronaći pločicu preko USB porta. Testira se svaki port zasebno uz pomoć biblioteke serialport i modula ReadyParser koji pregledava testirani port da li je poslan string 'Arduino' koji je pločica namještena da šalje u tom trenutku. Ako je tijekom testiranja porta pronađen taj „delimiter“, postavlja se port arduina na taj port. Postavljeno je vrijeme čekanja prije izvršavanja zatvaranja porta zbog problema koji mogu nastati ako se port ne zatvori, a aplikacija krene dalje.

```
function findArduino() {
      SerialPort.list().then(com_ports => {
            com_ports.forEach(function (port) {
                  var Test = new SerialPort({ path: port.path, baudRate:
115200 });
                  var readyparser = Test.pipe(new ReadyParser({ delimiter:
'Arduino' }));

                  Test.on('open', function () {
                        console.log('Trying serial port: ', port.path);
                  });

                  readyparser.on('ready', function () {
                        console.log('Arduino found at ', Test.path);
                        arduino_com_port = Test.path;
                        return;
                  });

                  setTimeout(function () {
                        Test.close(function () {
                              console.log(port.path, ' closed.');
                        });
                  }, 4000);
```

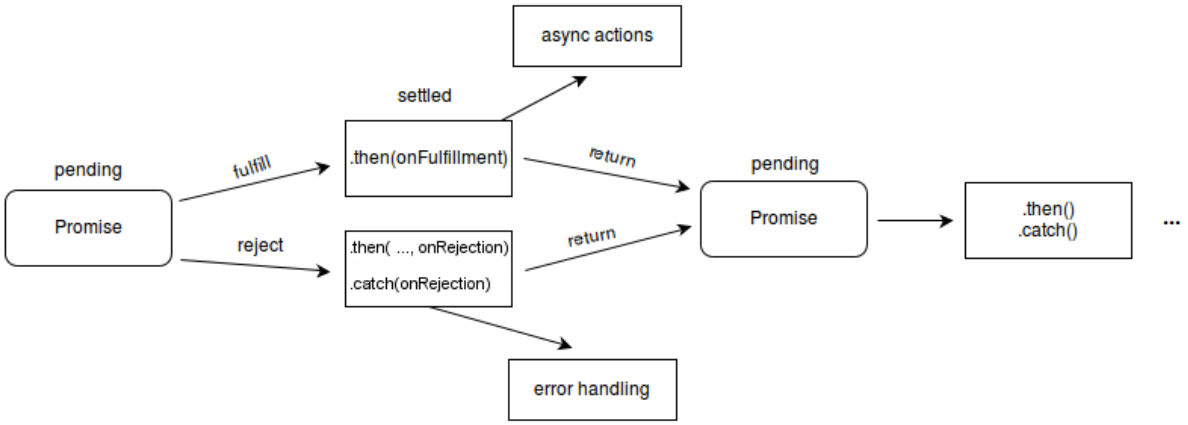
```
            });
      });
}
```

Nakon te funkcije, izvršava se funkcija ifFound() sa jednim argumentom 'again' koji će uskoro biti bolje pojašnjen. Također je umetnuto i vrijeme čekanja prije izvršavanja te funkcije od par sekundi kako bi se dala šansa funkciji findArduino() da nađe port na kojem je mikro kontroler. Unutar funkcije provjerava se da li je on pronađen i ako je pokreće se funkcija begin također sa parametrom 'again' koji joj prosljeđujemo i koji će biti pojašnjen bolje unutar te funkcije. Ako mikro kontroler nije pronađen, korisnik dobiva poruku koja ga moli da ponovno pokrene aplikaciju sa kontrolerom spojenim u port.

```
function IfFound(again) {
      if (arduino_com_port != 'comX') {
            begin(again);
      }
      else {
            console.log("Arduino board not found, please restart the
application with arduino plugged in USB port.")
            const keypress = async () => {
                  process.stdin.setRawMode(true)
                  return new Promise(resolve => process.stdin.once('data',
() => {
                        process.stdin.setRawMode(false)
                        resolve()
                  }))
            };
            (async () => {

                  console.log('Press any key to continue...')
                  await keypress()
            })().then(process.exit)
      }
}
```

Kako bi se bolje razumio ostatak programskoga koda, potrebno ja razumjeti JavaScript objekt obećanje ili eng. Promise. Obećanje je zamjena za vrijednost koja nije nužno poznata kada je obećanje stvoreno. Omogućuje se pridruživanje rukovatelja s eventualnom vrijednošću uspjeha ili razlogom neuspjeha asinkrone akcije. To asinkronim metodama omogućuje vraćanje vrijednosti poput sinkronih metoda; umjesto da odmah vrate konačnu vrijednost, asinkrona metoda vraća obećanje da će dati vrijednost u nekom trenutku u budućnosti. Obećanje je riješeno ako je ispunjeno ili odbijeno, ali ne na čekanju.



Slika 17. Promise objekt (developer.mozilla, 2022.)

Početak funkcije begin() otvara novu vezu sa USB portom na kojem je pločica te uz pomoć modula ReadLineParser biblioteke serialport i objekta Promise čeka na odgovor sa pločice koji će biti sadržan unutar varijable Wifi. Nakon što aplikacija dobije odgovor, pokreće se lokalna funkcija CheckConnection() koja isprva provjerava sadržaj Wifi varijable. Ako je mikro kontroler poslao da je povezan na WiFi, pokreće se ponovo jedan Promise koji čeka IP adresu s kojom je mikro kontroler povezan na WiFi. Kada je dobije, pokušava se ponovo povezati. No, ako je mikro kontroler poslao da nije povezan na WiFi, izvršavaju se ulančani objekti Promise koji čekaju na korisnikov unos novog WiFi SSID i lozinke. Nakon što korisnik te podatke unese, aplikacija ih šalje pločici te ponovo čeka na novu IP adresu sa pločice ako se uspije povezati sa mrežom. No, postavljeno je i vrijeme čekanja za prekid čekanja ako se u tom određenom vremenu pločica ne uspije povezati sa WiFi mrežom ili se je pločica povezala na krivu mrežu. Ako se dogodi prekid, korisnika se pita uz pomoć funkcije ReadAfterNoResponse() da li želi ponovno upisati SSID i lozinku za WiFi. Uz podatak da li je pločica povezana na WiFi, provjerava se i varijabla 'again'. Već prije spomenuta varijabla ima funkciju u slučaju da je pločica spojena na krivi WiFi, a korisnik želi upisati nove podatke da aplikacija ne ode u odjeljak koji čeka novu IP adresu s kojom neće biti moguće se povezati jer računalo nije na toj mreži. Ako je aplikacija dobila važeću IP adresu koja se provjerava pomoću funkcije ValidateIPaddress(), IP adresa se sprema u prijašnje spomenutu datoteku saved_ip.json koja se stvara ako već ne postoji.

Funkcija ReadAfterNoResponse() daje jednostavan upit uz pomoć Promise i readline.question te ovisno o korisnikovu unosu izvršava ponovan unos podataka za WiFi, izlazi iz aplikacije ili šalje poruku pogreške o pogrešnom unosu.

```
function begin(again) {
    var Wifi;
    var ArduinoPort = new SerialPort({ path: arduino_com_port, autoOpen:
false, baudRate: 115200 });
```

```
     const lineStream = ArduinoPort.pipe(new ReadlineParser({ Delimiter:
'\r\n' }));
     ArduinoPort.open(function () {
          console.log('Starting communication with Arduino on ' +
arduino_com_port + '.\nData rate: ' + ArduinoPort.baudRate);
          ArduinoPort.on('close', PortClosed);
          ArduinoPort.on('error', showError);
          const promise = new Promise(resolve => {
               lineStream.on('data', (data) => {
                    Wifi = data.trim();
                    resolve(data);
               });
          });
          promise.then(() => {
               CheckConnection();
          });



     });
     function ReadAfterNoResponse() {
          return new Promise((resolve, reject) => {
               readline.question('Try entering ssid and password again?
(y/n):  ', function (yesno) {
                    if (yesno == 'y' || yesno == 'Y') {
            again = true;
                         resolve(CheckConnection());
                    }
                    else if (yesno == 'n' || yesno == 'N')
                         resolve(process.exit());
                    else if (yesno != 'y' || yesno != 'Y' || yesno !=
'n' || yesno != 'N') {
                         reject("Wrong input.");
                         readAfterNoResponse()
                              .catch((error) => console.log('error',
error));
                    }
               });
          })
               .catch((error) => console.log('error', error));
     }
     function PortClosed() {
          console.log('Serial port closed.');


     }
     function showError(error) {
          console.log('Serial port error: ' + error);
     }
     function sendToSerial(data) {
          ArduinoPort.write(data);
     }


     function CheckConnection() {
          if (Wifi == "WifiOK" && again == false) {
               console.log("Waiting for new IP address...");
               return new Promise(resolve => {
                    lineStream.on('data', (data) => {
                         if (ValidateIPaddress(data.trim()) == true) {
                              if (socket.connecting == false) {
                                   console.log("New IP address: " +
data.trim())
```

```
                                        host = data.trim();
                                        fs1.writeFileSync("./saved_ip.json",
JSON.stringify(data.trim()));
                                        resolve(data, console.log("Retrying
connection..."),
                                             socket.connect(port, host),
setTimeout(function () { ArduinoPort.close(); }, 3000));
                                   }
                              }
                         });
                    });
               }
               else if (Wifi == "WifiNotOK" || again == true) {
                    again = false;
                    return new Promise(resolve => {
                         readline.question('Input new WiFi SSID: ', function
(ssid) {
                              resolve(sendToSerial(ssid + '\r'));
                              return new Promise(resolve => {
                                   readline.question('Input new WiFi password:
', function (pass) {
                                        resolve(sendToSerial(pass + '\n'),
console.log("WiFi info sending to Arduino..."));
                                        return new Promise((resolve, reject)
=> {
                                             console.log("Waiting for new IP
address from Arduino...");
                                             lineStream.on('data', (data) => {
                                                  if
(ValidateIPaddress(data.trim()) == true) {
                                                       if (socket.connecting
== false) {
                                                            console.log("New
IP address: " + data.trim());
                                                            host =
data.trim();

fs1.writeFileSync("./saved_ip.json", JSON.stringify(data.trim()));
                                                            resolve(data,
console.log("Retrying connection..."),

socket.connect(port, host), setTimeout(function () { ArduinoPort.close();
}, 3000));
                                                       }
                                                  }
                                             });
                                             setTimeout(() => {
                                                  if (isConnected == false &&
socket.connecting == false) {
                                                       reject("Waiting for IP
address timed out.");
                                                       console.log("If the LED
strip has a red loading bar, arduino is not connected to WiFi.\n If it has
a purple loading bar, the WiFi is connected, but your PC is not on the same
network.");
                                                       ReadAfterNoResponse();
                                                  }
                                             }, 18000);
                                        })
                                             .catch((error) =>
console.log('error', error));
```

```
                                    });
                               })
                          });
                     });
                }
           }
}
```

Funkcija ValidateIpadress() :

```
function ValidateIPaddress(ipaddress) {
     if (/^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)$/.test(ipaddress)) {
          return (true)
     }
     return (false)
}
```

Također je potrebno spomenuti što se dogodi kada se pozove socket.connect() i kada se ostvari događaj socket.close() koji se pozove nakon određenog vremenskog perioda u kojem socket.connect() se nije uspio povezati na IP adresu i port.

Nakon svakog socket.connect(), aplikacija samo obavještava korisnika da je uspješno povezana sa pločicom te su aplikacija i pločica spremne za integraciju sa igrom.

```
socket.on('connect', () => {
      console.log(`Connected to Arduino board`);
      console.log(`On local port: ${socket.localPort}\n`);
      console.log(`Ready for connection with dota2.`);
      isConnected = true;

});
```

Kada se izvrši socket.close(), obavještava korisnika da nije bilo uspješno povezivanje sa pločicom te ga pita uz pomoć funkcije read() da li želi ponovo upisati podatke za WiFi. Ovaj isječak koda postoji da bi se riješili slučajevi kada je pločica povezana sa WiFi mrežom, no aplikacija se ne može povezati na pločicu (pločica je na različitoj mreži od računala).

```
socket.on('close', () => {
     isConnected = false;
     console.log(`Cannot connect to board.`);
     console.log("If the LED strip has a red loading bar, arduino is not
connected to WiFi.\n If it has a purple loading bar, the WiFi is connected,
but your PC is not on the same network.");
     read();
     function read() {
          return new Promise((resolve, reject) => {
               readline.question('Try entering ssid and password again?
(y/n): ', function (yesno) {
                    if (yesno == 'y' || yesno == 'Y') {
                         if (arduino_com_port != 'comX') {
                              var again = true;
```

```
                              resolve(IfFound(again));
                        }
                        else {
                              findArduino();
                              var again = true;
                              setTimeout(function () { IfFound(again); },
7000);
                        }
                  }
                  else if (yesno == 'n' || yesno == 'N')
                        resolve(process.exit());
                  else if (yesno != 'y' || yesno != 'Y' || yesno !=
'n' || yesno != 'N') {
                        reject("Wrong input.");
                        read()
                              .catch((error) => console.log('error',
error));
                  }
            });
      })
            .catch((error) => console.log('error', error));
  }
});
```

Konekcija je napravljena na način da za prvu konfiguraciju pločica mora biti spojena sa USB kabelom, a bilo koji drugi put aplikacija i pločica će raditi samo uz zahtjev da je pločica spojena na napajanje od 5V.

5.2.2. Spremanje konfiguracijske datoteke za Dota 2 GSI

Dota 2 GSI ne radi ako u određenoj mapi unutar Dota 2 instalacijske mape ne postoji određena konfiguracijska datoteka. Zato se na početku izvođenja programa gleda da li postoji ta datoteka unutar te mape kada se pozove funkcija saveConfig(). Prvo se stvara varijabla za spremanje podataka koji trebaju biti u toj datoteci, te se dohvaća putanja na računalu gdje je Dota 2 instalirana uz pomoć funkcije getDotaPath(), Ako datoteka ne postoji, stvara se .cfg datoteka na dohvaćenoj putanji sa writeFile().

```
function saveConfig() {
    let cfg = VDF.stringify({
        'dota2-gsi Configuration': {
            'uri': "http://localhost:3000/",
            'timeout': 5,
            'buffer': 0.1,
            'throttle': 0.1,
            'heartbeat': "30.0",
            'data':
            {
                'buildings': 1,
                'provider': 1,
                'map': 1,
                'player': 1,
                'hero': 1,
```

```
                'abilities': 1,
                'items': 1,
                'draft': 1,
                'wearables': 1
            },
            'auth':
            {
                'token': "hello1234"
            }
        }
    })
    getDotaPath(function (pth) {
        fs.exists(path.join(pth, GSIFILENAME), function (exists) {
            if (!exists) {
                fs.writeFile(path.join(pth, GSIFILENAME), cfg)
                console.log("Config saved to dota2 installation folder.");
            }
            else
                console.log("Config file already exists in dota2 folder.");
        });
    })
}
```

Funkcija getDotaPath() se sastoji od dvije druge funkcije od kojih jedna dohvaća iz registra put do .vdf datoteke koja govori na kojim tvrdim diskovima se sve nalaze biblioteke programa Steam preko kojeg je instalirana Dota 2. Druga funkcija čita tu .vdf datoteku i vraća sve puteve do svih biblioteka na svakom tvrdom disku. Nakon te dvije funkcije uz pomoć pathExists() provjerava se postoji li Dota 2 u svim tim mapama. Ako postoji, funkcija getDotaPath() vraća putanju do te mape.

```
function getDotaPath(cb) {
     getLibraryFoldersVDFPath(function (vdfPath) {
          getLibraryFolders(vdfPath, function (libraries) {
               libraries.push(path.join(path.dirname(vdfPath), 'common'))
               libraries.forEach(function (pth, index) {
                    let cfgPath = path.join(pth, 'dota 2 beta', 'game',
'dota', 'cfg', 'gamestate_integration')
                    fs.pathExists(cfgPath, function (err, exists) {
                         if (!err && exists) {
                              cb(cfgPath)
                         }
                    })
               })
          })
     })
}
```

Funkcija getLibraryFoldersVDFpath() pretražuje registar za ključ \\Software\\Valve\\Steam u kojemu su sadržane vrijednosti putanje prema .vdf datoteci. Potrebna je biblioteka winreg.

```
function getLibraryFoldersVDFPath(cb = function () { }) {
     if (os.platform() === 'win32') {
          let regKey = new Registry({
```

```
                hive: Registry.HKCU,
                key: '\\Software\\Valve\\Steam'
          })
          regKey.values(function (err, items) {
                let r = false
                if (err) {
                      console.log('ERROR: ' + err)
                }
                else {
                      for (var i = 0; i < items.length && !r; i++) {
                            if (items[i].name === REG) {
                                  r = path.join(items[i].value, WINSTEAMLIB)
                            }
                      }
                }
                cb(r)
          })
     }
     else {
          let vdfpath
          if (os.platform() === 'darwin') {
                vdfpath = path.join(HOMEDIR, 'Library/Application
Support/Steam/steamapps/libraryfolders.vdf')
          }
          else {
                vdfpath = path.join(HOMEDIR,
'.local/share/Steam/steamapps/libraryfolders.vdf')
          }
          cb(vdfpath)
          return vdfpath
     }
}
```

getLibraryFolders() čita .vdf datoteku sa putanje koju je dobila od prijašnje funkcije te vraća Steam mape na svakom tvrdom disku ako postoje.

```
function getLibraryFolders(vdfPath, cb) {
     let folders = []
     try {
          fs.readFile(vdfPath, { encoding: 'utf8' }, function (err, data)
{
                if (!err) {
                      let vdfdata = VDF.parse(data)
                      if ('LibraryFolders' in vdfdata) {
                            vdfdata = vdfdata['LibraryFolders']
                            let key
                            for (key in vdfdata) {
                                  if (!isNaN(key)) {

folders.push(path.join(vdfdata[key].path, 'steamapps', 'common'))
                                  }
                            }
                      }
                }
                cb(folders)
          })
     }
     catch (err) {
          cb(folders)
```

```
    }
}
```

5.2.3. Dohvaćanje podataka iz igre

Sljedeći isječak koda prikazuje skupinu slušatelja koji čekaju događaje iz Dota 2 klijenta na računalu. Kada Dota2 GSI server koji je na početku programa pozvan dobije događaj da se je povezao novi Dota2 klijent s njim, uključuju se i ostali slušatelji koji iz klijenta dobavljaju podatke o stanju igre. To mogu biti razni podaci kao npr. ukupan broj smrti, broj novčića, trenutno stanje heroja (karaktera u igri), oprema heroja, herojeve moći i njihovo trenutno stanje, itd. Za izradu ovog uređaja potrebni su hero:alive koji nam govori da li je heroj mrtav, hero: health_percent i mana_percent za stanje života i resursa mane heroja te hero:respawn_seconds koji govori koliko je preostalo sekundi da se heroj oživi.

Nakon što se je dogodio određeni događaj unutar igre, tj. jedan od spomenutih, aplikacija preko WiFi konekcije šalje pločici podatke kako bi ih ona mogla prikazati na RGB LED traci. Također je i rukovan krajnji slučaj ako se aplikacija poveže sa pločicom, a već je počela runda unutar igre na način da pri svakom povezivanju se šalju osnovni podaci samo jedanput kako bi se LED traka mogla uspostaviti na trenutne vrijednosti. Isto tako, ako korisnik trenutno nije u igri, šalju se prazni podaci kako bi se traka ugasila.

```
server.events.on('newclient', function (client) {
    client.on('player:activity', function (activity) {
        if (activity == 'playing') console.log("Game started!");
        console.log(activity);
    });

    client.on('hero:health_percent', function (alive) {
        rawRequest = client.gamestate.hero.health_percent.toString();
        socket.write(rawRequest + "H \r\n");
    });

    client.on('hero:mana_percent', function (alive) {
        rawRequest = client.gamestate.hero.mana_percent.toString();
        socket.write(rawRequest + "M \r\n");
    });
    client.on('hero:alive', function (alive) {
        if (alive == true) {
            rawRequest = "1";
            socket.write(rawRequest + "D \r\n");
            rawRequest =
client.gamestate.hero.health_percent.toString();
            socket.write(rawRequest + "H \r\n");
            rawRequest = client.gamestate.hero.mana_percent.toString();
            socket.write(rawRequest + "M \r\n");
        }
        else if (alive == false) {
            rawRequest = "0";
```

```
                socket.write(rawRequest + "D \r\n");
            }
        });

        client.on('hero:respawn_seconds', function (respawn_seconds) {
            rawRequest = respawn_seconds.toString();
            socket.write(rawRequest + "S \r\n");
        });



        if (client != 0) {
            if (client.gamestate.hero &&
client.gamestate.hero.health_percent && client.gamestate.hero.mana_percent
&& client.gamestate.hero.alive) {
                if (client.gamestate.hero.alive == true) {
                    rawRequest = "1";
                    socket.write(rawRequest + "D \r\n");
                    rawRequest =
client.gamestate.hero.health_percent.toString();
                    socket.write(rawRequest + "H \r\n");
                    rawRequest =
client.gamestate.hero.mana_percent.toString();
                    socket.write(rawRequest + "M \r\n");
                }
                if (client.gamestate.hero.alive == false) {
                    rawRequest = "0";
                    socket.write(rawRequest + "D \r\n");


                }
            }
        }

        setInterval(function () {
            if (client != 0) {
                if (!client.gamestate.hero) {
                        socket.write("0" + "H \r\n");
                        socket.write("0" + "M \r\n");
                        socket.write("1" + "D \r\n");
                        socket.write("0" + "S \r\n");
                }
            }
        }, 10 * 1000);


});
```

5.3. Izrada programskog koda za mikro kontroler

5.3.1. Kreiranje WiFi servera

Kao prvo, moraju se deklarirati sljedeće globalne varijable:

```
String ssid_serial;
String password_serial;
char ssid[20] = "";
char password[20] = "";
#define PIN 14
String Info = "";
String Realssid;
String Realpassword;
int isAlive;
int Max_Percentage = 100;
float Max_Percentage_float = 100;
float SecondsToRespawn;
float SecondsToRespawnStarting;
float SecondsRatio = 0;
bool countdown = false;
bool Send = true;
WiFiServer server(80);
String request;
int Health_Percent;
int Mana_Percent;
int Index;
Adafruit_NeoPixel strip = Adafruit_NeoPixel(29, PIN, NEO_GRB +
NEO_KHZ800);
```

Kako bi se nakon konfiguracije sa USB kabelom pločica mogla sama spojiti na WiFi, bilo je potrebno omogućiti spremanje podataka za WiFi na mikro kontroler. To je ostvareno uz pomoć biblioteke EEPROM.h koja služi za spremanje podataka na EEPROM. Svaki put kada se piše ili čita sa EEPROM-a mora se pozvati EEPROM.begin() te EEPROM.end() kada se završi sa radom.

```
void loadCredentials() {
  EEPROM.begin(512);
  EEPROM.get(0, ssid);
  EEPROM.get(0 + 50, password);
  EEPROM.end();
  Realssid = String(ssid);
  Realssid.trim();
  Realpassword = String(password);
  Realpassword.trim();
}
void saveCredentials() {
  EEPROM.begin(512);
  strcpy(ssid, ssid_serial.c_str());
  strcpy(password, password_serial.c_str());
  EEPROM.put(0, ssid);
  EEPROM.put(0 + 50, password);
  EEPROM.commit();
  EEPROM.end();
}
```

Nadalje, svaki arduino program ima dvije glavne funkcije: setup() i loop(). Setup() će se pokrenuti samo jedanput, dok se loop nakon setupa izvršava cijelo vrijeme dok mikro kontroler ima napajanja. Unutar funkcije setup(), pokrećemo serijsku vezu i uključujemo LED traku uz pomoć biblioteke Adafruit_Neopixel. Nakon toga stoji dio gdje se pločica spaja na WiFi. Poziva se funkcija da pročita zadnje spremljene podatke o WiFi - u sa EEPROM - a te se od spajamo sa WiFi - a za slučaj da postoji veza. Sve dok veza nije uspostavljena, šalje se preko serijske veze identifikator kako bi aplikacija mogla naći mikro kontroler te obavijest da WiFi nije spojen. Isto tako nakon slanja ta dva stringa, poziva se funkcija StandingBy() koja prikazuje na LED traci učitavanje crvenom bojom, što znači da nije uspješno spajanje s WiFi. Nakon toga, ako postoje nadolazeći podaci sa serijske veze, učitavaju se podaci ili u varijablu za SSID ili u varijablu za lozinku od WiFi mreže. Nakon spremanja tih podataka, program se vraća na učitavanje podataka koji su bili spremljeni i ponovnog pokušaja povezivanja.

```
void setup() {
  Serial.begin(115200);
  strip.begin();
  strip.setBrightness(64);
  strip.show();
label:
  loadCredentials();
  WiFi.disconnect();
  WiFi.begin(Realssid, Realpassword);
  while (WiFi.status() != WL_CONNECTED) {
    Send = true;
    Serial.println("Arduino");
    Serial.println("WifiNotOK");
    StandingBy(strip.Color(255, 0, 0));
    while (Serial.available()) {
      char aChar = Serial.read();
      Info += aChar;
      if (aChar == '\r') {
        ssid_serial = Info;
        Info = "";
      }
      if (aChar == '\n') {
        password_serial = Info;
        Info = "";
        saveCredentials();
        Send = true;
        goto label;
      }
    }
  }
  server.begin();
}
```

Funkcija StandingBy() jednostavno simulira na LED traci učitavanje kao što bi se vidjelo tijekom neke softverske instalacije. Tri LED diode se pale u određenim intervalima dok ne dođu do kraja trake.

```
void StandingBy(uint32_t color) {
  for (uint16_t i = 0; i < strip.numPixels(); i++) {
    strip.clear();
    for (uint16_t j = i; j < i + 3 && j < strip.numPixels(); j++) {
      strip.setPixelColor(j, color);
      delay(25);
    }
    strip.show();
  }
  strip.clear();
  strip.show();
}
```

Ako se je pločica uspješno povezala sa mrežom, pokreće se loop() te dio za slučaj da je potrebna ponovna konfiguracija ako se je pločica spojila sa mrežom na kojoj računalo nije. Ponovo se gleda da li ima nadolazećih podataka na serijskoj vezi te se oni učitavaju u konfiguraciju. U isto vrijeme ponavlja se simuliranje učitavanja i slanje identifikatora te obavijesti o povezanosti, ali ovaj puta sa ljubičastom bojom koja označava da je pločica spojena na WiFi. Ako je pločica ponovo dobila nove podatke za WiFi, poziva se ponovno funkcija setup() radi spajanja s novim SSID i lozinkom.

```
void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    WiFiClient client = server.available();
    while (Serial.available()) {
      char aChar = Serial.read();
      Info += aChar;
      if (aChar == '\r') {
        ssid_serial = Info;
        Info = "";
      }
      if (aChar == '\n') {
        password_serial = Info;
        Info = "";
        saveCredentials();
        Send = false;
        Serial.end();
        setup();
      }
    }

    if (!client) {
      Serial.println("Arduino");
      Serial.println("WifiOK");
      if (Send == true) {
        Serial.println(WiFi.localIP());
      }
      StandingBy(strip.Color(191, 64, 191));
      return;
    }
    while (client.connected()) {
```

Programski kod unutar petlje while(client.connected()) nalazi se u idućem poglavlju.

```
            }

      delay(1);
   }
   else {
      Serial.end();
      setup();
   }
}
```

Ako se veza sa WiFi mrežom izgubi, pločica odlazi nazad u setup() i šalje ponovo podatke aplikaciji da WiFi nije povezan.

5.3.2. Primanje podataka iz aplikacije

Cijeli vremenski period u kojemu je aplikacija spojena na WiFi server pločica sluša za nadolazeće podatke. Svaki podatak se čita do kraja i ovisno o identifikatoru (u ovom slučaju određena slova) ti podaci se spremaju u varijable kako bi se mogli prenijeti vizualno preko LED trake. Podaci koji se spremaju su postotak života, postotak mane, stanje heroja (živ ili mrtav) te sekunde do oživljavanja ako je heroj mrtav. Ako je heroj živ, podaci o životnim i mana bodovima se obrađuju uz pomoć funkcija SetHealth() i SetMana(). No ako je heroj mrtav, LED traka to vizualno također pokazuje uz pomoć funkcije Death(). Svaka od ovih funkcija je u for petlji koja je određena mjestom vizualizacije na traci. Također sve tri funkcije primaju dva argumenta – jedan koji govori funkciji trenutni broj indeksa LED diode za provjeru (i) i jedan koji je pomoćna varijabla za formulu kojom se pale određena led svjetla. Jedina razlika između funkcije za smrt i funkcija za životne i mana bodove je ta da se omjer početnih sekundi i trenutnih sekundi mora računati jer se ne može dobiti iz igre.

```
while (client.connected()) {
       if (client.available()) {
          strip.clear();
          char c = client.read();
          request += c;
          if (c == '\n') {
             if (request.indexOf('H') != -1) {
                Index = request.indexOf('H');
                Health_Percent = request.substring(0, Index).toInt();
                request = "";
             }
             else if (request.indexOf('M') != -1) {
                Index = request.indexOf('M');
                Mana_Percent = request.substring(0, Index).toInt();
                request = "";
             }
             else if (request.indexOf('D') != -1) {
                Index = request.indexOf('D');
                isAlive = request.substring(0, Index).toInt();
```

```
                request = "";
            }
            else if (request.indexOf('S') != -1) {
                Index = request.indexOf('S');
                SecondsToRespawn = request.substring(0, Index).toInt();
                if (countdown == false) {
                    SecondsToRespawnStarting = SecondsToRespawn;
                    countdown = true;
                }
                request = "";
            }
            if (isAlive == 1) {
                SecondsToRespawnStarting = 0;
                countdown = false;
                for (uint16_t i = strip.numPixels() / 2, sum = 1; i <
strip.numPixels(); i++, sum++) {
                    SetHealth(i, sum);
                }

                for (uint16_t i = strip.numPixels() / 2, sum = 1; i > 0;
i--, sum++) {
                    SetMana(i, sum);
                }
            }
            if (isAlive == 0 && SecondsToRespawn != 0 &&
SecondsToRespawnStarting != 0) {
                SecondsRatio = (SecondsToRespawn /
SecondsToRespawnStarting) * 100;
                for (uint16_t i = 0, sum = 1; i < strip.numPixels();
i++, sum++) {
                    Death(i, sum);
                }
                if (SecondsToRespawn == 0) {
                    for (uint16_t i = 0; i < strip.numPixels(); i++) {
                        strip.setPixelColor(i, strip.Color(169, 169, 169));
                    }
                }
            }

            strip.show();

        }
    }

}
```

5.3.3. Prikazivanje podataka

Formula za određivanje indeksa led diode koja će se upaliti ili ugasiti je ista za sve tri funkcije i radi na način da dijeli maksimalni postotak (koji je 100) na onoliko koliko je potrebno LED dioda za prikazivanje podatka (što je količina / 2 za životne i mana bodove, a cijela traka za smrt) i uz to pomoćna varijabla sum određuje granice intervala između tih LED dioda unutar kojih se provjerava jesu li životni bodovi, mana bodovi ili sekunde do oživljavanja veći od prve granice i manji ili jednaki od druge granice. Kroz for petlju koja je postavljena prije tih funkcija, prolazi se kroz svaki interval i ako je podatak prešao granicu, upaliti će se ili ugasiti određen broj LED dioda nakon što se pozove strip.show() na kraju petlje u kojoj se provjeravaju podaci.

```
if (Health_Percent != 0 && Health_Percent > Max_Percentage -
((Max_Percentage / (strip.numPixels() / 2))*sum) && Health_Percent
<= (Max_Percentage - ((Max_Percentage / ((strip.numPixels() / 2)) *
(sum - 1)))))
```

Funkcija SetHealth():

```
void SetHealth(uint16_t LEDnum, int sum) {
  if (Health_Percent != 0 && Health_Percent > Max_Percentage -
((Max_Percentage / (strip.numPixels() / 2))*sum) && Health_Percent <=
(Max_Percentage - ((Max_Percentage / ((strip.numPixels() / 2)) * (sum -
1)))) {
    for (uint16_t i = strip.numPixels(); i > LEDnum; i--) {
      strip.setPixelColor(i, strip.Color(0, 255, 0));
    }
  }
  if (Health_Percent == 0) {
    strip.clear();
  }
}
```

Funkcija SetMana():

```
void SetMana(uint16_t LEDnum, int sum) {
  if (Mana_Percent != 0 && Mana_Percent > Max_Percentage -
((Max_Percentage / (strip.numPixels() / 2))*sum) && Mana_Percent <=
(Max_Percentage - ((Max_Percentage / (strip.numPixels() / 2)) * (sum -
1)))) {
    for (uint16_t j = 0; j < LEDnum; j++) {
      strip.setPixelColor(j, strip.Color(0, 0, 255));
    }
  }
  if (Mana_Percent == 0) {
    strip.clear();
  }
}
```

Funkcija Death():

```
void Death(uint16_t LEDnum, int sum) {
  if (SecondsRatio != 0 && SecondsRatio > Max_Percentage_float -
((Max_Percentage_float / (strip.numPixels()))*sum) && SecondsRatio <=
(Max_Percentage_float - ((Max_Percentage_float / (strip.numPixels())) *
(sum - 1)))) {
    for (uint16_t i = 0; i < LEDnum; i++) {
      strip.setPixelColor(i, strip.Color(169, 169, 169));
    }
  }
}
```

5.4. Distribucija

Sama aplikacija na računalu je spakirana pomoću biblioteke pkg. Ta biblioteka omogućuje pakiranje cijelog Visual Studio projekta napravljenog u Node.js unutar jedne .exe datoteke.

6. Zaključak

Iz ovoga rada možemo zaključiti da internet stvari može biti namijenjen i za zabavu. No, proći će još neko vrijeme prije nego se prosječni igrač računalnih igara krene svakidašnje koristiti IoT uređajima koji su povezani sa igrom. Moguće je da će nas svijet virtualne i proširene stvarnosti uvesti u novo doba, ali prvo treba pričekati da uređaji koji omogućuju igranje takvih igara (sa mogućnošću spajanja na internet) postanu više pristupačni prosječnom kupcu. Zasada su ti uređaji jednostavno preskupi, a većina igara nisu bolja od običnih računalnih igara.

Dok ne dođe ta nova era, možemo stvarati uređaje slične ovome koji je bio izrađen unutar rada. Dapače, nije se morala koristiti RGB LED traka za prikazivanje podataka iz igre. Ima pregršt različitih mogućnosti koje mogu biti pozvane nakon određenih događaja unutar igre kao što su zvučni signali, kontroliranje servo motora, daljnje spajanje sa različitim web servisima, itd. Samo je potrebno imati komponente koje se mogu spojiti sa mikro kontrolerom.

7. Budući rad

Kod same izrade aplikacije na računalu mogao sam birati između dvije biblioteke koje omogućavaju implementaciju Dota 2 GSI – Node.js i C#. Odabrao sam Node.js jer sam kroz fakultetsko obrazovanje i više nego dovoljno naučio programski jezik C# te sam imao volje naučiti nešto novo. Programski kod za arduino mi nije bio ništa novo pošto je programski jezik C++. Sve u svemu, bilo mi je zanimljivo raditi na ovoj temi, a pogotovo na uređaju. Ovisno o vremenu i mogućnostima, definitivno bih pokušao napraviti u slobodno vrijeme nešto slično ili pak sasvim drugačije za neku drugu računalnu igru i preporučio bih ovakav projekt svim entuzijastima računalnih igara.

Popis literature

1. Beyers, J. (2019.). IoT as Part of Evolution of Gaming Industry.

Preuzeto 27. 5. 2022. s https://hackernoon.com/iot-in-evolution-of-gaming-industry-fo1t32tz

2. Bahga, A. i Madisetti, V. (2014.). Internet Of Things A Hands On Approach. Preuzeto 27. 5. 2022. s

https://www.kngac.ac.in/elearning-portal/ec/admin/contents/4_18KP2CS07_2021012902234424.pdf

3. Salazar, J. i Silvestre, S. (2017.). Internet Of Things. Prag: Češko tehničko sveučilište u Pragu – Fakultet elektroinženjerstva

4. Evans, D. (2011.). The Internet of Things - How the Next Evolution of the Internet Is Changing Everything. Preuzeto 28. 5. 2022. s

https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

5. Benford, S., Magerkurth, C. i Ljungstrand, P. (2005.). Bridging the physical and digital in pervasive gaming. Preuzeto 28. 5. 2022. s

https://www.researchgate.net/publication/220427210_Bridging_the_physical_and_digital_in_pervasive_gaming

6. Blast Theory (2003.). Can You See Me Now?. Preuzeto 28. 5. 2022. s https://www.blasttheory.co.uk/projects/can-you-see-me-now/

7. Muskan (2021.) 4 Applications of IoT in Gaming Industry. Preuzeto 28. 5. 2022. s https://www.analyticssteps.com/blogs/4-applications-iot-gaming-industry

8. Johansen, P. (2018.). IoT – Based Pervasive Game Framework. Preuzeto 28. 5. 2022. s

https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2565102/18477_FULLTEXT.pdf?sequence=1

9. Flatner, A., Oystein Bjorkeng, H. Nicolausson, S., Rimolsronning, A. i Syvertsen, A. (2021.). Exploring the Possibilities of Using Serious Gaming and IoT to Improve Students' Habits. Preuzeto 29. 5. 2022. s

https://folk.idi.ntnu.no/baf/eremcis/2021/Group10.pdf

10. Calvo - Morata, A., Alonso - Fernandez, C., Freire, M. i Martinez - Ortiz, I. (2021.). Creating awareness on bullying and cyberbullying among young people: Validating the effectiveness and design of the serious game Conectado. Preuzeto 29. 5. 2022. s

https://www.researchgate.net/publication/348889187_Creating_awareness_on_bullying_and_cyberbullying_among_young_people_Validating_the_effectiveness_and_design_of_the_serious_game_Conectado

11. Wouters, P., van Nimwegen, C., van Oostendorp, H. i van der Spek, E. (2013.). A meta - analysis of the cognitive and motivational effects of serious games. Preuzeto 29. 5. 2022. s

https://www.researchgate.net/publication/263936571_A_Meta-Analysis_of_the_Cognitive_and_Motivational_Effects_of_Serious_Games

12. Kosmides, P., Demestichas, K., Adamopolou, E., Koutsouris, N. Oikonomidis, Y. i De Luca, V. (2018.). InLife: Combining Real Life with Serious Games using IoT preuzeto 29. 5. 2022. s

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8490434

13. Techaheadcorp (2020.). Evolution of Internet of Things (IoT): Past, present and future. Preuzeto 29. 5. 2022. s https://www.techaheadcorp.com/knowledge-center/evolution-of-iot/

14. Zubair Khan, M., Alhazmi, O., Awais Javed, M., Ghandorh, H. i Aloufi, K. (2021.). Reliable Internet of Things: Challenges and Future Trends. Preuzeto 29. 5. 2022. s https://www.mdpi.com/2079-9292/10/19/2377/pdf

15. IBM (2020.). Application Programming Interface (API). Preuzeto 29. 5. 2022. s https://www.ibm.com/cloud/learn/api

16. Pip Boy [Slika] (2021.) cdn.mos [Sa interneta] Preuzeto 30. 5. 2022. s https://cdn.mos.cms.futurecdn.net/kxSsHqqgZPbf9uBRrRFdJR-970-80.jpg

17. Magerkurth, C., Mandryk, R., Cheok, A. i Nilsen, T. (2005.). Pervasive games: bringing computer entertainment back to the real world. Preuzeto 10. 6. 2022. s https://www.researchgate.net/publication/220686485_Pervasive_games_bringing_computer_entertainment_back_to_the_real_world

18. Cunningham, S. i Henry, J. (2020.). Augmenting Virtual Spaces: Affective Feedback in Computer Games. Preuzeto 20. 6. 2022. s https://www.researchgate.net/publication/338839427_Augmenting_Virtual_Spaces_Affective_Feedback_in_Computer_Games

19. Henry, J., Tang, S., Hanneghan, M. i Carter, C. (2017.). A Measure of Student Engagement for Serious Games and IoT. Preuzeto 22. 6. 2022. s https://www.researchgate.net/publication/316828333_A_Measure_of_Student_Engagement_for_Serious_Games_and_IoT

20. Introduction to Node.js. (2022.). Nodejs.dev [Na internetu] Preuzeto 1. 7. 2022. s https://nodejs.dev/en/learn

21. Promise. (2022.). developer.mozilla [Na internetu] Preuzeto 5. 7. 2022. s https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

22. Simões, M., Bernardes, M., Barros, F. i Castelo - Branco, M. (2018.). Virtual travel training for autism spectrum disorder: proof-of-concept interventional study.

23. Maines, C.L., Llewellyn - Jones, D., Tang, S. i Zhou, B. (2015.). A cyber security ontology for BPMNsecurity extensions.

24. Marcinkowski, D. (2022.). VRChat [Slika] Preuzeto 30. 6. 2022. s https://blog.readyplayer.me/vrchat-3d-avatars-from-photo-success-story/

25. Promise objekt [Slika] (2022.). Preuzeto 2. 7. 2022. s https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Popis slika