SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 3852

# PRIKAZ UMJETNE INTELIGENCIJE STABLOM PONAŠANJA

Valentin Berger

Zagreb, lipanj 2015.

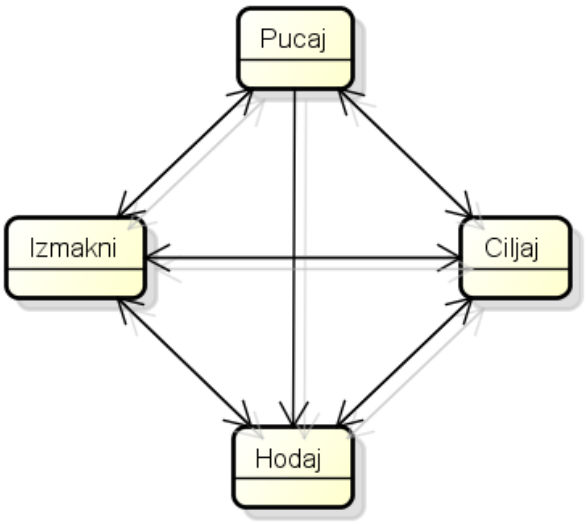
# Zahvala

Hvala mentoru prof. dr. sc. Domagoju Jakoboviću što mi je dao punu slobodu biranja i obrade teme završnog rada.

# Sadržaj

# Uvod

Stablo ponašanja (engl. *Behavior Tree*) danas je jedan od poznatijih načina prikaza umjetne inteligencije u video igrama i robotici gdje je potrebno stvoriti umjetno inteligentnog agenta ili robota koji se mora snalaziti u nekoj virtualnoj ili stvarnoj okolini. Ponašanja takvih agenata u igrama mogu biti vrlo ekspresivna i vrlo kompleksna zato što za svako posebno stanje okoline agent ili robot mora „znati“ kako se ponašati. No, prije nego li se je za prikaz takve inteligencije koristilo stablo




*Slika 1*

ponašanja, koristili su se konačni automati (Slika 1). Svako različito ponašanje (engl. *behavior*) bi se opisalo jednim stanjem, a stanje okoline koja aktivira određeno stanje bi se opisalo funkcijama prijelaza između stanja. Taj način ostvarivanja inteligencije je brz i efikasan, ali pati od nekoliko nepremostivih mana koje se javljaju pri izradi kompleksnijih aplikacija (Pereira 2014.):

- **Sposobnost održavanja** (engl. *maintainability*) – kako broj različitih ponašanja raste, tj. kako inteligencija postaje kompleksnija, to je dodavanje novih ili izmjena starih ponašanja teže jer se moraju izmijeniti i svi uvjeti prelazaka između novih i starih stanja.
- **Razumljivost** (engl. *scalability*) – što više stanja (ponašanja) automat ima, to je teže razumjeti inteligenciju kao cjelinu, a stoga i teže dodavanje novih ponašanja.

- **Ponovna uporaba** (engl. *reusability*) – jednom kada se stanja/ponašanja dodaju u automat ona postaju strogo povezana (engl. *hard coded*) s drugim stanjima i agentom, stoga je teško upotrijebiti ta stanja/ponašanja u drugim objektima i aplikacijama.
- **Mogućnost paralelizacije** (engl. *parallelization*) – jako je teško izvoditi stanja u automatu paralelno i sinkronizirati ga, a da se ne pojave potpuni zastoji (engl. *deadlocks*).

Zbog svih navedenih razloga javlja se potreba da se ponašanja agenata/robota strukturiraju na drugačiji način odnosno, pojavila se je potreba za stablom ponašanja. Uporaba stabla ponašanja za ostvarenje umjetne inteligencije pojavila se je u industriji igara, ali je kasnije formalizirana od znanstvenika ponajprije zbog široke uporabe u robotici. Za razliku od konačnih automata stablo ponašanja rješava prethodno navedene nedostatke (Pereira 2014.):

- **Sposobnost održavanja** – za razliku od konačnih automata prelasci između različitih ponašanja nisu u uvjetima nego u strukturi stabla, stoga je dodavanje novih i oduzimanje starih ponašanja puno jednostavnije jer se ne mora dirati ostatak stabla.
- **Razumljivost** – možda i najveća prednost Stabla Ponašanja je ta što su lako razumljiva i jer se svako podstablo može gledati kao posebno stablo tj. pod inteligencija. Zbog toga je i puno lakše grafički prikazati stablo jer se može „razbiti“ na sitnije dijelove.
- **Ponovna uporaba** – zbog toga što je svako ponašanje/čvor neovisno o ostatku stabla, ponovna uporaba takvog čvora u drugim objektima i aplikacijama je jednostavna.
- **Mogućnost paralelizacije** – mogućnost paralelizacije je relativno laka jer se paralelizam odvija lokalno u podstablu te zbog toga imamo kontrolu nad time koliko i gdje će se odvijati paralelizam.

Ovo su samo neke od prednosti stabla ponašanja, a detaljna analiza će se prikazati u narednim poglavljima.

# 1. Stablo ponašanja

## 1.1. Ponašanje stabla

Kao i svi drugi algoritmi tako je i stablo ponašanja kroz svoju kratku povijest imalo različite implementacije i unaprjeđenja. Zbog toga se stabla ponašanja mogu podijeliti na ona prve generacije i na ona druge generacije. Struktura stabla je i u jednoj i u drugoj generaciji jednaka, ali se bitno razlikuju u načinu interpretacije. Stabla druge generacije se mogu podijeliti na one „pogonjene“ podatcima (engl. *data driven*) i one „pogonjene“ na događaje (engl. *event driven*) (Champandard 2012.). Riječ „pogonjen“ u ovome kontekstu označava način na koji se stablo ponašanja izvodi. Ovaj rad će se uglavnom bazirati na stablu ponašanja „pogonjenom“ na događaje, ali će se i objasniti kako funkcionira stablo prve generacije.

Stablo prve generacije je zamišljeno kao skup unutarnjih čvorova čijom se interpretacijom izvršavaju listovi koji čine uvijete (engl. *conditions*) i akcije (engl. *actions*). Kako je u svim video igrama zamišljeno da svaki agent ima svoj „otkucaj sata“ nakon kojeg se osvježava njegovo unutarnje stanje, tako je i stablo ponašanja zamišljeno da se može osvježavati svaki određeni period. Zbog toga se nad korijenom periodično generira signal (engl. *tick*) koji se zatim propagira kroz stablo i osvježava ga, tj. osvježava ponašanje agenta. Kako je već prethodno rečeno stablo je podijeljeno na unutarnje i vanjske čvorove (listove). Uloga unutarnjih čvorova je odrediti koji vanjski čvor će se izvršiti. Kada se neki vanjski čvor u jednom otkucaju izvrši njegova je uloga da vrati povratnu informaciju roditelju o svojemu stanju, a koju roditelj zatim propagira nazad prema korijenu stabla, ovisno o tome kojeg je tipa neki unutarnji čvor. Jedna od glavnih stanja (engl. *status*) u kojima se mogu nalaziti listovi, a koji se koriste u svim inačicama stabla ponašanja su:

- **`Uspjeh`** (engl. *success*) – označava da se je vanjski čvor uspješno izvršio. U slučaju da je vanjski čvor uvjet to znači da je uvjet točan.

- **`Neuspjeh`** (engl. *failure*) – označava da se je vanjski čvor nije uspješno izvršio, a u slučaju da je vanjski čvor uvjet to znači da je uvjet netočan.

- **`Izvršava se`** (engl. *running*) – označava da se vanjski čvor još izvršava, tj. da se u sljedećem otkucaju opet treba izvršiti.

- **`Nevažeći`** (engl. *invalid*) – označava da čvor ili nije inicijaliziran ili da se je dogodila neka druga pogreška.

Navedena stanja vrijede i za unutarnje čvorove, no njihova je uloga da obrade stanje djeteta i da odrede vlastito stanje koje se zatim propagira prema korijenu stabla. Na koji način neki unutarnji čvor obrađuje stanje djeteta ovisi o njegovom tipu, tj. tipu unutarnjeg čvora.

## 1.2. Tipovi čvorova stabla

Čvorovi stabla ponašanja mogu se podijeliti u tri kategorije: dekoratere (engl. *decorator*), kompozite (engl. *composite*) i listove (engl. *leaf*). Svi čvorovi se koriste preko zajedničkog sučelja koje ima metodu za obradu tog čvora. Budući da čvorovi stabla obogaćuju samo stablo (inteligenciju) jasno da su prve dvije kategorije dobile ime po oblikovnim obrascima (engl. *design patterns*), a o čemu će se više govoriti u narednim poglavljima.

### 1.2.1. Listovi

Listovi stabla se mogu podijeliti u dvije podkategorije: akcije (engl. *actions*) i uvjeti (engl. *conditions*).

Akcije su čvorovi u kojima se obavljaju neke operacije nad agentima (npr. agent se kraće prema cilju) i takvi čvorovi poprimaju jedan od tri statusa: `uspjeh` (npr. agent je uspješno došao do cilja), `neuspjeh` (npr. agent nije uspio doći do cilja), `izvršava se` (npr. agent se još uvijek kreće prema cilju).

Uvjeti su pak čvorovi u kojima se provjerava neki uvjet nad agentom, tj. nekom njegovom varijablom. Uvjeti mogu poprimiti status: `uspjeh` (npr. agent je pronašao put), `neuspjeh` (npr. agent nije pronašao put).

Te dvije kategorije čvorova definiraju ponašanja agenta, ali ne i redoslijed i trenutak izvođenja za što su odgovorni unutarnji čvorovi stabla (dekorateri i kompoziti).

### 1.2.2. Dekorateri

Dekorateri stabla mogu imati samo jedno dijete i njihova uloga je da manipuliraju povratnim stanjem svojeg djeteta, a da zatim na temelju tog stanja odrede stanje koje će propagirati svojem roditelju. Dekorateri prema ulogama mogu biti razni, ali jedni od osnovnih su: `Ponavljač` (engl. *Repeater*), `Izmjenjivač` (engl. *Inverter*), `Onaj koji uvijek uspijeva` (engl. *Succeeder*), `Ponavljač dok ne uspije` (engl. *Until fail*). Stanja u kojima se mogu nalaziti ovi čvorovi su neki od navedenih u prethodnom poglavlju, ovisno o tipu dekoratera.

`Ponavljač` je tip unutarnjeg čvora koji izvršava svoje dijete određeni broj puta ili beskonačno, tj. nakon što aktivira podčvor (dijete) i nakon što mu se vrati odgovor on opet aktivira taj isti podčvor. Ako se `Ponavljač` izvršava određeni broj puta onda nakon određenog broja ponavljanja postavlja (prosljeđuje) svoje stanje na onu vrijednost koju je zadnje primio (očitao) od svog djeteta, tj. prosljeđuje to stanje svojem roditelju.

`Izmjenjivač` radi na principu da ukoliko od svog djeteta primi (očita) stanje `uspjeh`/`neuspjeh`/`izvršava se`, svome roditelju proslijedi (ili postavi svoje stanje) `neuspjeh`/`uspjeh`/`izvršava se`.

`Čvor koji uvijek uspijeva` (engl. *Succeeder*) za bilo koju vrijednost stanja svoga djeteta, svome roditelju vraća stanje `uspjeh`.

`Ponavljač` dok ne uspije (engl. *Until fail*) funkcionira slično kao ponavljač, ali on se izvršava sve dok ne očita (primi) `neuspjeh` od svog djeteta.

Kombinacijom navedenih čvorova mogu se ostvariti i druge funkcionalnosti (npr. kombinacijom `Izmjenjivača` i čvora `Koji uvijek uspijeva` možemo dobiti čvor `Koji nikada ne uspijeva` itd.)

### 1.2.3. Kompoziti

Za razliku od dekoratera kompoziti mogu imati jedno ili više djece, a slično kao i kod dekoratera oni manipuliraju stanjima (povratnim) svoje djece i zatim na temelju tih stanja utvrđuju svoje stanje koje zatim propagiraju svojem roditelju. Također, tipovi kompozita mogu bit svakakvi, no neki osnovni su: `Selektor` (engl. *Selector*), `Slijed` (engl. *Sequence*), `Usporednik` (engl. *Parallel*). Oni se razlikuju po načinu i redoslijedu izvođenja svoje djece.

`Selektor` (još se zna nazivati i prioritet (engl. *Priority*)) ima ulogu da „okida“ svoju djecu redom (s lijeva na desno) sve dok neko dijete ne vrati stanje `uspjeh` kada i sam selektor vraća stanje `uspjeh` svome roditelju. Ukoliko niti jedno dijete ne vrati stanje `uspjeh` odnosno sva djeca vrate stanje `neuspjeh` tada selektor svome roditelju vraća stanje `neuspjeh`. Način na koji funkcionira selektor je slično logičkoj operaciji ILI u logičkoj formuli (gdje je formula selektor, a djeca čine literale) jer je potreban jedan točan uvjet/literal (stanje `uspjeh` kod djece), a da i sama formula bude istinita. Također, čim se pronađe dijete koje ima stanje `uspjeh` odmah se prekida daljnje izvođenje ostale djece i selektor poprima vrijednost `uspjeh` (engl. *Short-circuit evaluation*).

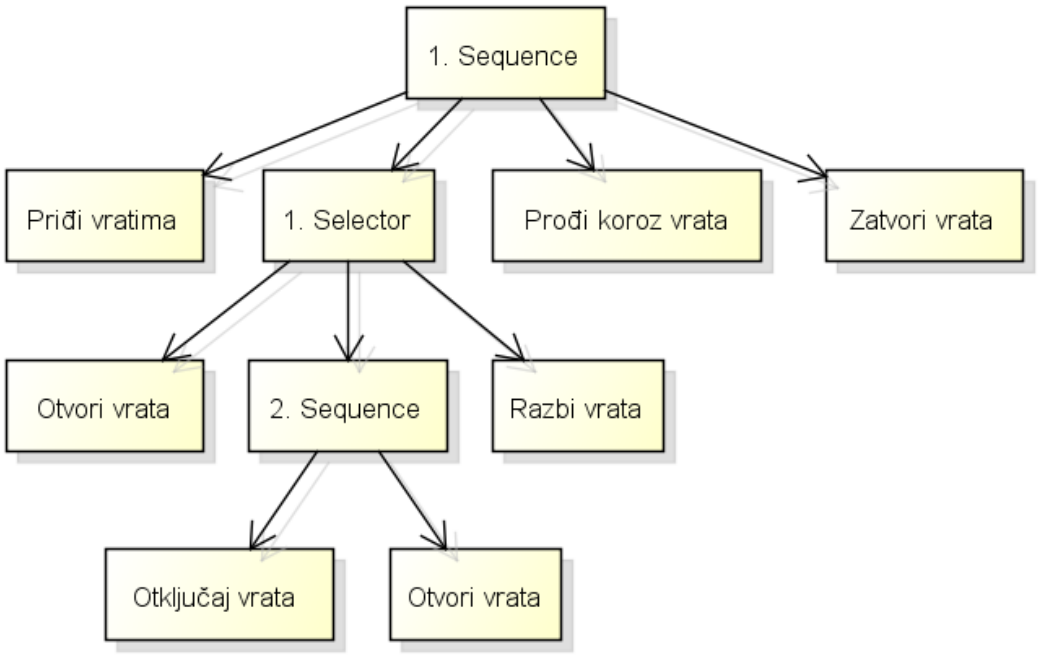
`Slijed` (engl. *Sequence*) funkcionira suprotno od `Selektora`. On izvršava čvorove djece redom sve dok neko dijete ne vrati stanje `neuspjeh` kada i sam `Slijed` vraća svome roditelju stanje `neuspjeh`. Ako obradi svu svoju djecu onda se njegovo stanje postavlja u `uspjeh` i završava se njegovo izvršavanje. Kao što se `Selektor` može poistovjetiti s logičkim operatorom ILI, tako se `Slijed` može poistovjetiti s logičkim operatorom I.

`Usporednik` (engl. *Parallel*) služi za uvođenje paralelizacije u stablo ponašanja tako što tako što u istom „otkucaju“ pokreće izvođenje dvoje ili više čvorova djece, a zatim čeka na njihovo izvršenje. `Usporednik` vraća `uspjeh` ako je broj djece koja vrate stanje `uspjeh` veći od neke konstante S, vraća `neuspjeh` ako je broj djece koja vrate stanje `neuspjeh` veći od neke konstante F, a inače vraća stanje `izvršava se`. Konstante S i F su parametri `Usporednika` i mogu biti drugačiji u svakoj instanci `Usporednika`.

Postoje i drugi tipovi kompozita kao npr. oni koji imaju nasumičan redoslijed izvođenja djece.

## 1.3. Stablo pogonjeno na događaje

U ovom završnom radu biti će prikazana i opisana implementacija stabla ponašanja pogonjenog na događaje (engl. *event driven Behavior Tree*). Zašto pogonjeno na događaje? Za primjer ćemo uzeti dio inteligencije nekog agenta koju koristi pri prolasku kroz vrata (Slika 2). Pretpostavimo pritom da prilazak vratima, otključavanje i razbijanje vrata traje dva „otkucaja“, a sve ostale akcije jedan otkucaj, da su vrata kroz koja agent mora proći zaključana i da agent ima ključ za otključati vrata. Kao što je već navedeno stablo ponašanja prve generacije zasniva se na tome da se pri svakom „okidanju“ stabla na temelju stanja čvorova odredi koji se čvor treba izvršiti, a zatim da se povratna informacija stanja čvora koji se je izvršio propagira nazad prema korijenu stabla, i tako svaki „otkucaj sata“. Sljedeća tablica (Tablica 1) opisuje način izvođenja stabla ponašanja prve generacije.




*Slika 2 (Simpson 2014.)*

*Tablica 1*

| R.b. „otkucaja“ | Čvor koji se izvršava | Opis | Stanje (povratno) |
|---|---|---|---|
| 1. | 1. Sequence | Odabire se prvo dijete „Priđi vratima“ kao aktivno | Izvršava se |
| 1. | Priđi vratima | Agent prilazi vratima, ali još nije prišao. | Izvršava se |
| 1. | 1. Sequence | Budući da se aktivno dijete još izvršava, završava se otkucaj. | Izvršava se |
| 2. | 1. Sequence | Ponovno se odabire zadnje aktivno dijete „Priđi vratima“ | Izvršava se |
| 2. | Priđi vratima | Agent je uspješno prišao vratima | Uspijeh |
| 2. | 1. Sequence | Odabire se sljedeće dijete „1. Selector“ kao aktivno | Izvršava se |
| 2. | 1. Selector | Odabire se prvo dijete „Otvori vrata“ kao aktivno | Izvršava se |
| 2. | Otvori vrata | Vrata se ne mogu otvoriti zato što su zaključana | Neuspjeh |
| 2. | 1. Selector | Odabire se sljedeće dijete „2. Sequence“ | Izvršava se |
| 2. | 2. Sequence | Odabire se prvo dijete „Otključaj vrata“ kao aktivno | Izvršava se |
| 2. | „Otključaj vrata“ | Vrata se počinju otključavati | Izvršava se |
| 2. | 2. Sequence | Budući da se aktivno dijete još izvršava, ništa se ne događa | Izvršava se |

| | | | |
|---|---|---|---|
| 2. | 1. Selector | Budući da se aktivno dijete još izvršava, ništa se ne događa | Izvršava se |
| 2. | 1. Sequence | Budući da se aktivno dijete još izvršava, završava se otkucaj | Izvršava se |
| 3. | 1. Sequence | Ponovno se odabire zadnje aktivno dijete „1. Selector“ | Izvršava se |
| 3. | 1. Selector | Ponovno se odabire zadnje aktivno dijete „2. Sequence“ | Izvršava se |
| 3. | 2. Sequence | Ponovno se odabire zadnje aktivno dijete „Otključaj vrata“ | Izvršava se |
| 3. | Otključaj vrata | Vrata se uspješno otključavaju | Uspjeh |
| 3. | 2. Sequence | Odabire se sljedeće dijete „Otvori vrata“ kao aktivno | Izvršava se |
| 3. | Otvori vrata | Vrata se uspješno otvaraju | Uspjeh |
| 3. | 2. Sequence | Budući da su sva djeca uspješno izvršena, uspješno je završen i čvor | Uspjeh |
| 3. | 1. Selector | Budući da je zadnje aktivno dijete uspješno završeno, uspješno je završen i čvor | Uspjeh |
| 3. | 1. Sequence | Odabire se sljedeće dijete „Prođi kroz vrata“ kao aktivno | Izvršava se |
| 3. | Prođi kroz vrata | Agent uspješno prolazi kroz vrata | Uspjeh |
| 3. | 1. Sequence | Odabire se sljedeće dijete „Zatvori vrata“ | Izvršava se |

| | | | |
|---|---|---|---|
| | | kao aktivno | |
| 3. | Prođi kroz vrata | Agent uspješno prolazi kroz vrata | Uspjeh |
| 3. | 1. Sequence | Budući da su sva djeca uspješno izvršena, uspješno je završen i čvor | Uspjeh |

Kao što je vidljivo iz primjera u stablu ponašanja prve generacije, u svakom otkucaju stabla ima puno nepotrebnog kretanja po stablu prilikom traženja zadnjeg aktivnog čvora, ali i nepotrebnog kretanja prema korijenu stabla kada se neka akcija mora izvršavati duže od jednog „otkucaja“. Taj problem dolazi sve to više do izražaja što je dubina stabla ponašanja veća i može uzrokovati usko grlo (engl. *bottleneck*) u aplikacijama. Kako bi se izbjegao taj problem razvijeno je stablo ponašanja pogonjeno na događaje, a koje spada u stabla ponašanja druge generacije. Prvi navedeni problem se rješava na sljedeći način. Umjesto da se pri svakom otkucaju obilazi stablo kako bi se našao zadnje aktivni list, aktivni listovi se drže u redu za izvršavanje te se odmah prilikom novog otkucaja iz reda uzimaju aktivni listovi, izvršavaju se i ako se trebaju opet izvršiti (akcija traje dulje od jednog „otkucaja“) stavljaju se na kraj reda te se pri novom otkucaju ponovno izvršavaju. Drugi problem, tj. problem obavještavanja roditelja nakon što se neki čvor uspješno ili neuspješno izvrši rješava se uz pomoć oblikovnog obrasca Promatrač (engl. *Observer*). To znači da svaki roditelj koji krene izvršavati mora promatrati svoje dijete i čekati da se ono izvrši (uspješno ili neuspješno), a dijete je dužno nakon što se izvrši obavijestiti svojeg roditelja o tome. Na taj način se izbjegava nepotrebno obilaženje stabla. Više o implementaciji stabla ponašanja pognojenog na događaje biti će opisano u narednim poglavljima. Na sljedećoj tablici (Tablica 2) za usporedbu je riješen isti prethodni primjer, ali sa stablom pognojenim na događaje. Već i na ovom kratkom primjeru je vidljivo da stablo ponašanja pogonjeno na događaje obavlja manji broj operacija nego stablo prve generacije. Na dubljim stablima ta razlika je još puno osjetljivija.

*Tablica 2*

| R.b. „otkucaja“ | Čvor koji se izvršava | Opis | Stanje (povratno) |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1. | 1. Sequence | Odabire se prvo dijete „Priđi vratima“ kao aktivno | Izvršava se |
| 1. | Priđi vratima | Agent prilazi vratima, ali još nije prišao | Izvršava se |
| 2. | Priđi vratima | Agent uspješno prilazi vratima | Uspjeh |
| 2. | 1. Sequence | Odabire se sljedeće dijete „1. Selector“ kao aktivno | Izvršava se |
| 2. | 1. Selector | Odabire se prvo dijete „Otvori vrata“ kao aktivno | Izvršava se |
| 2. | Otvori vrata | Vrata se ne mogu otvoriti zato što su zaključana | Neuspjeh |
| 2. | 1. Selector | Odabire se sljedeće dijete „2. Sequence“ | Izvršava se |
| 2. | 2. Sequence | Odabire se prvo dijete „Otključaj vrata“ kao aktivno | Izvršava se |
| 2. | „Otključaj vrata“ | Vrata se počinju otključavati | Izvršava se |
| 3. | Otključaj vrata | Vrata se uspješno otključavaju | Uspjeh |
| 3. | 2. Sequence | Odabire se sljedeće dijete „Otvori vrata“ kao aktivno | Izvršava se |
| 3. | Otvori vrata | Vrata se uspješno otvaraju | Uspjeh |
| 3. | 2. Sequence | Budući da su sva djeca uspješno izvršena, uspješno je završen i čvor | Uspjeh |
| 3. | 1. Selector | Budući da je zadnje aktivno dijete uspješno završeno, uspješno je završen | Uspjeh |

| | | | |
|---|---|---|---|
| | | i čvor | |
| 3. | 1. Sequence | Odabire se sljedeće dijete „Prođi kroz vrata“ kao aktivno | Izvršava se |
| 3. | Prođi kroz vrata | Agent uspješno prolazi kroz vrata | Uspjeh |
| 3. | 1. Sequence | Odabire se sljedeće dijete „Zatvori vrata“ kao aktivno | Izvršava se |
| 3. | Prođi kroz vrata | Agent uspješno prolazi kroz vrata | Uspjeh |
| 3. | 1. Sequence | Budući da su sva djeca uspješno izvršena, uspješno je završen i čvor | Uspjeh |

# 2. Programsko ostvarenje

U ovome poglavlju će biti opisan i dokumentiran način izrade samog stabla ponašanja pogonjenog na događaje i aplikacije uređivača (engl. *editor*) za izradu stabla ponašanja. Također će biti opisane tehnologije i pomoći koje su se pritom koristile.

## 2.1. Tehnologije

Za izradu stabla ponašanja i aplikacije za izradu stabla korištene su sljedeće tehnologije:

- C++11 unutar integriranog razvojnog sučelja Visual Studio 2013 – C++ je korišten za izvedbu algoritma stabla ponašanja tj. njegove funkcionalnosti. Također je korišten zbog svojih performansi, a što većina umjetnih inteligencija zahtijeva.

- Boost C++ - dodatne biblioteke za programski jezik C++

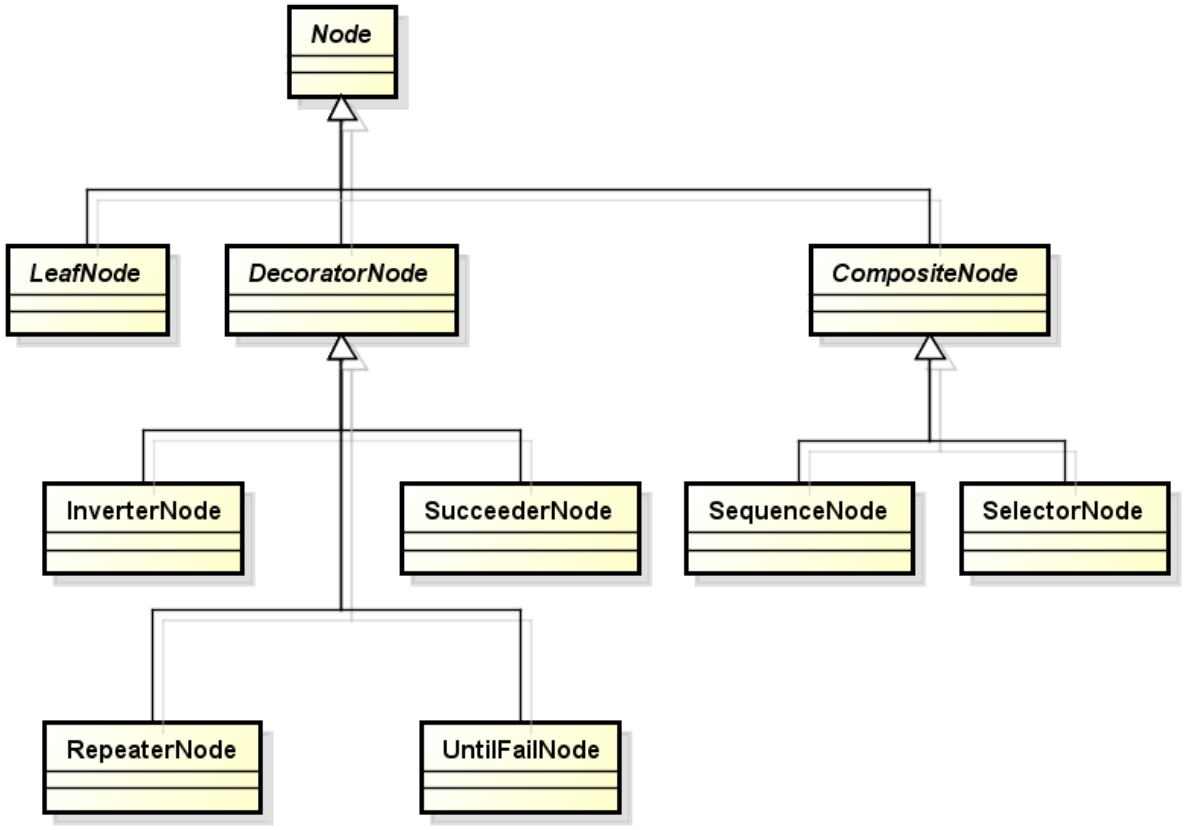
- C# .NET Framework 4.5 unutar integriranog razvojnog sučelja Visual Studio 2013 – C# je korišten pri izradi aplikacije uređivača stabla ponašanja. Korišten je isključivo zbog svoje jednostavnosti pri izradi aplikacija.

- SQL unutar integriranog razvojnog sučelja Visual Studio 2013 – korišten je za izradu lokalne baze podataka koju koristi aplikacija uređivač.

- Entity Framework – korišten je za jednostavnu komunikaciju između baze podataka i aplikacije uređivača stabla ponašanja.

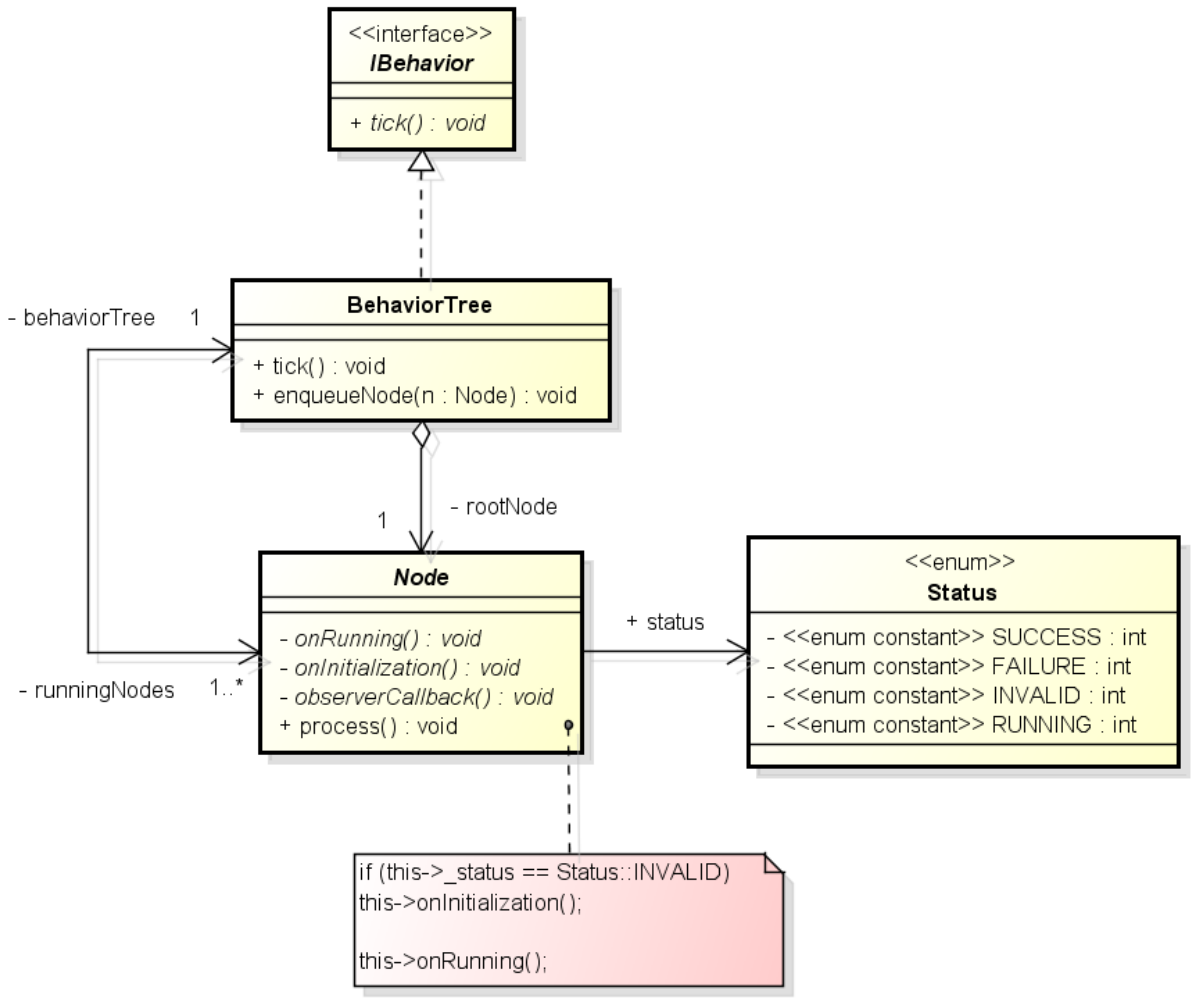
## 2.2. Izrada stabla ponašanja

Za izradu stabla ponašanja korišten je objektno orijentirani programski jezik C++ unutar integriranog razvojnog sučelja Visual Studio 2013. Kao vrsta projekta odabrana je statička biblioteka koja se može pronaći među predlošcima pri stvaranju novog projekta unutar Visual Studia. U prethodnim poglavljima navedeno je da stablo ponašanja čine različite vrste čvorova te se je stoga morala napraviti odgovarajuća hijerarhija svih čvorova stabla. Uz pomoć objektno orijentirane paradigme to je ostvareno kao što je prikazano na sljedećem dijagramu (Slika 3).




*Slika 3 – Prikaz hijerarhije čvorova stabla*

Apstraktni razred `Node` koji služi za komunikaciju s razredom `BehaviorTree` (Slika 4), a koji predstavlja stablo ponašanja. Razred `BehaviorTree` sadrži interni raspored izvođenja (listu) aktivnih čvorova, u koju se čvorovi mogu prijavljivati uz pomoć metode `enqueueNode`, i koju metoda `tick()` koristi za „tumačenje“ stabla. Na slici (Slika 5) se može se vidjeti kod metode `tick()`, a koja se izvršava prilikom „okidanja“ stabla. Kao što se može vidjeti na početku se u listu aktivnih čvorova (raspored) stavlja graničnik kako bi se odijelili čvorovi koji se trebaju izvršiti u trenutnom ciklusu od onih koji se trebaju izvršiti slijedeći ciklus. Nakon toga se obrađuje svaki čvor iz te liste pomoću metode `process()` razreda `Node` sve dok se ne naiđe na graničnik. Ako se neki čvor treba ponovno izvršiti, tj. njegovo izvršavanje traje dulje od jednog ciklusa on se stavlja na kraj liste (rasporeda) kako bi se ponovno izvršio slijedeći ciklus. Također, iz slike 4 može se vidjeti da razred `Node` ima okvirnu metodu (engl. *template method*) `process()` u kojoj je koriste apstraktne metode `onInitalization()` i `onRuning()`, a koje se implementiraju u nižim hijerarhijskim razinama čvorova, o čemu će biti riječ u nastavku.

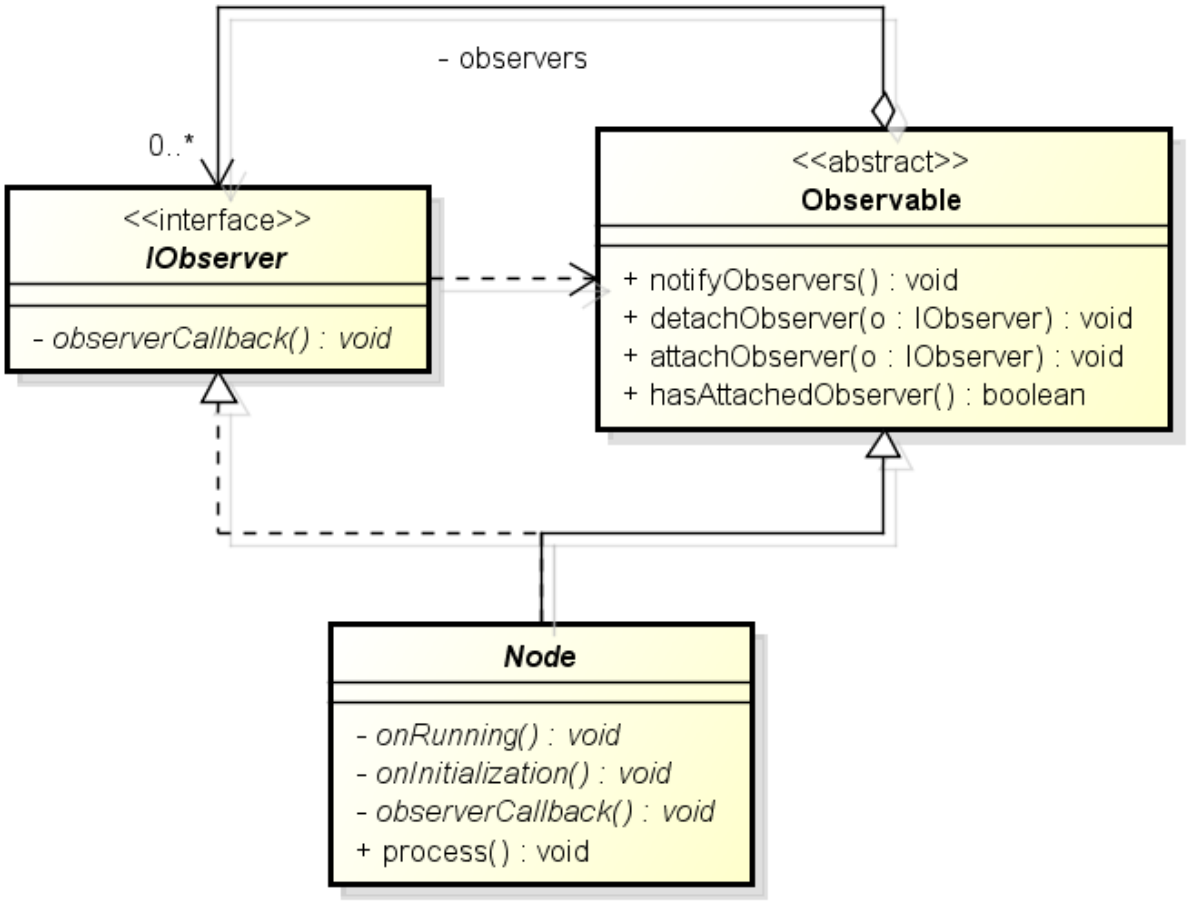



*Slika 4 – Prikaz interakcije razreda BehaviorTree i razreda Node, kao i prikaz primjene oblikovnog obrasca Okvirna Metoda na razred Node*

```
BehaviorTree& BehaviorTree::tick() {

	// Insert tree end-of-update marker to queue.
	_runningNodes.push_back(nullptr);

	// Update nodes from queue until marker is reached.
	while (this->processNextNode())
		continue;

	return *this;
}


bool BehaviorTree::processNextNode() {

	auto current = _runningNodes.front();
	_runningNodes.pop_front();

	// If end-of-update marker is reached stop processing.
	if (current == nullptr)
		return false;

	// Process next running node.
	current->process();

	// If node is succeeded or failed run observer if it exists, or enqueue node
	//again.
	if (current->getStatus() == Node::Status::FAILURE ||
	    current->getStatus() == Node::Status::SUCCESS)
		current->notifyObservers();
	else if (current->getStatus() == Node::Status::RUNNING)
		_runningNodes.push_back(current);

	return true;
}
```

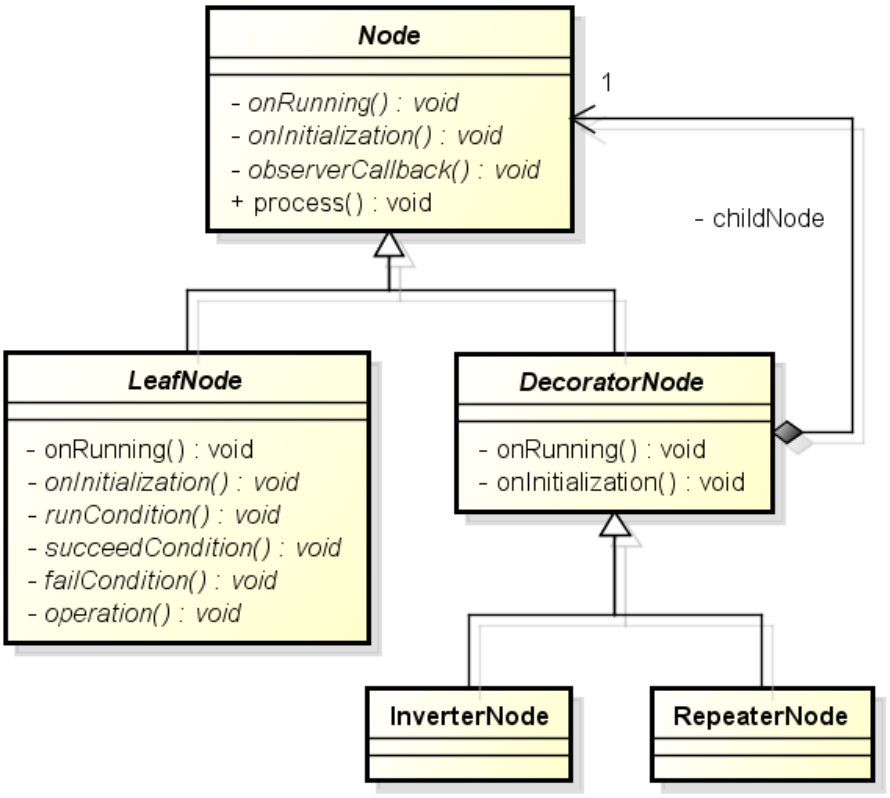
*Slika 5 – Prikaz koda koji se izvršava pri osvježavanju stabla*

Buduće da je uvjet za ostvarenje stabla ponašanja taj da svaki čvor roditelj mora osluškivati svoju djecu, ali i da djeca moraju obavijestiti roditelja kada su završili izvršavanje, korištenje oblikovnog obrazaca Promatrač (engl. *Observer*) je idealno rješenje tog problema. Budući da svi čvorovi (osim listova) mogu biti i promatrači i biti promatrani uvode se dva razreda iz kojih se razred `Node` izvodi, a to su sučelje `IObserver` i razred `Observable` (Slika 6). Metodu `observerCallback()` implementiraju tek konkretni razredi izvedeni iz razreda `DecoratorNode` i `CompositeNode` (`SequnceNode`, `InverterNode`, `RepeaterNode` i drugi) zato što te razrede razlikuje način na koji reagiraju kada ih dijete obavijesti da je gotovo s izvršavanjem.

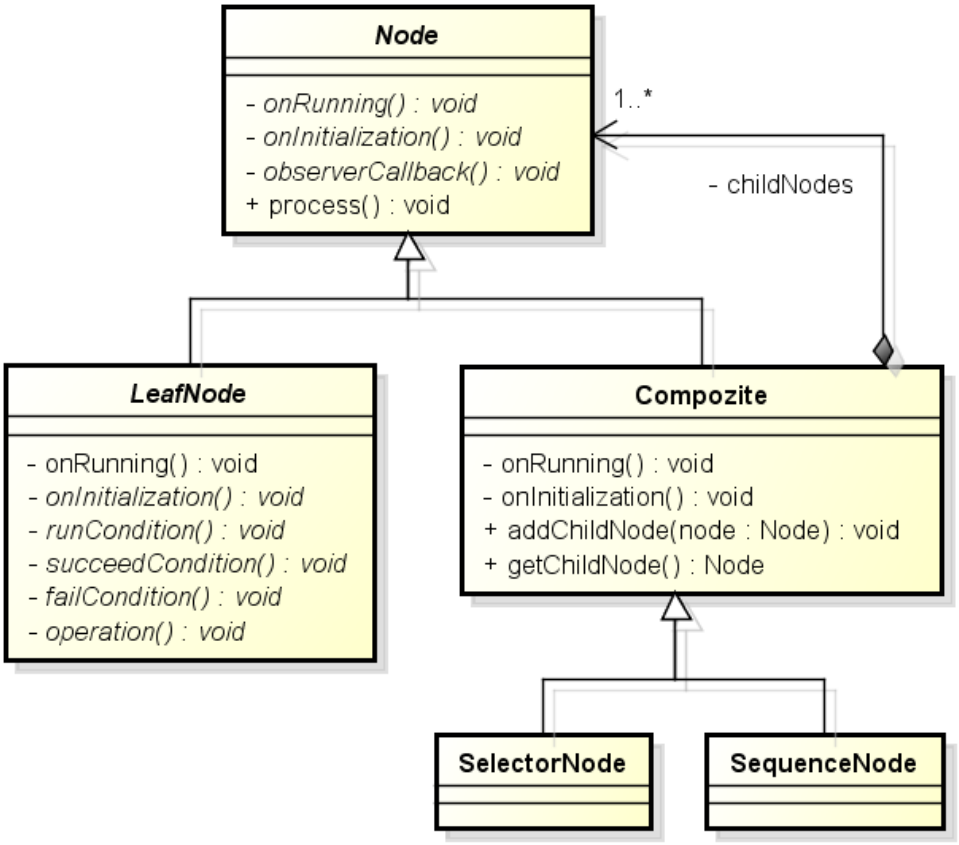



*Slika 6 – Slika prikazuje primjenu oblikovnog obrasca Promatrač na razred Node*

U prethodnim poglavljima smo spomenulo se je da postoje dva tipa čvorova Kompozit i Dekorater koji služe za ostvarivanje istoimenih oblikovnih obrasca. U kontekstu ovog projekta te čvorove predstavljaju razredi `DecoratorNode` (Slika 7) i `CompositeNode` (Slika 8). Može se vidjeti da se dekoracija, tj. kompozicija razreda `Node` obavlja preko metoda `onRunning()` i `onInitalization()`, a koje se koriste u prethodno navedenoj okvirnoj metodi `process()` razreda `Node`.

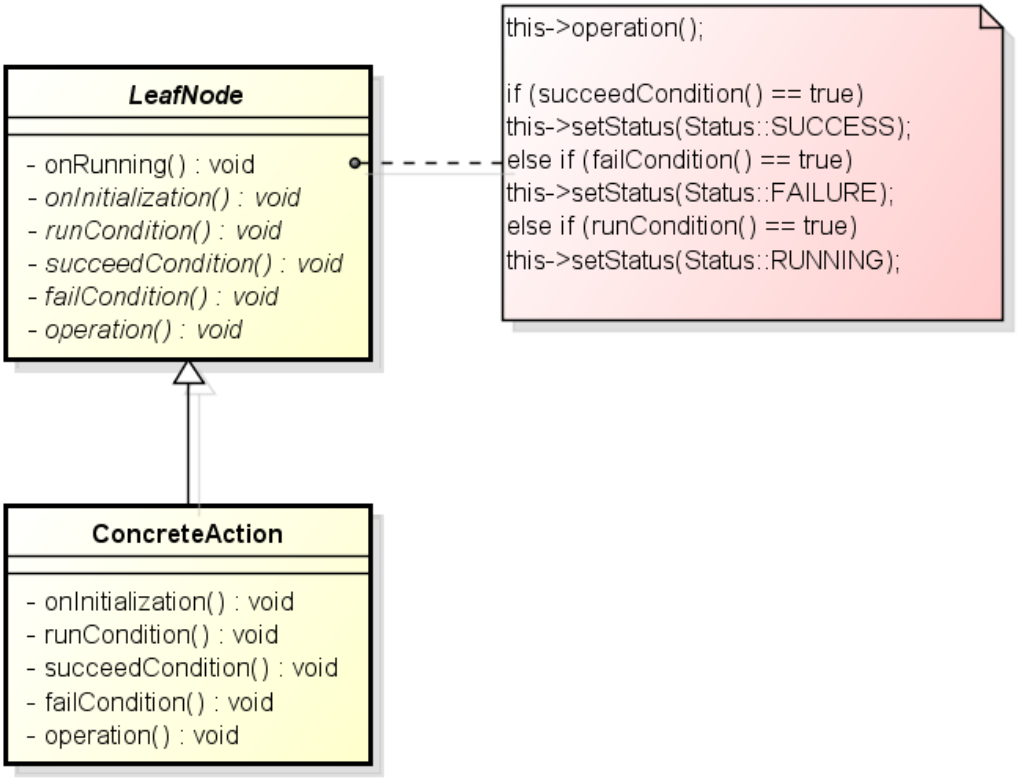



*Slika 7 - Prikaz primjene oblikovnog obrasca Dekorater na razred Node*




*Slika 8 - Prikaz primjene oblikovnog obrasca Kompozit na razred Node*

Na slikama 7 i 8 može se vidjeti da razred `LeadNode` uvodi nekoliko novih apstraktnih metoda (`runCondition()`, `succeedCondition()`, `failCondition()`, `operation()`). Te metode trebaju implementirati konkretni razredi (akcije agenta), a koristi ih okvirna metoda `onRunnng()` razreda `LeafNode` (Slika 9). Iz operacije `onRunning()` može se vidjeti da svaka konkretna akcija agenta može imati četiri dijela izvođenja: dio koda koji će se obavljati pri svakome izvođenju (metoda `operation()`) i jednu ili više operaciju uvjeta (`succeedCondition()`, `failCondition()`, `runCondition()`) ovisno o uspješnosti izvođenja uvjeta. Takav način izvođenja lista (konkretne akcije) stabla zamišljen je zato da korisniku omogući pisanje posebnog dijela koda za svaki uvjet kako bi se izbjegla potreba korisnika za poznavanjem enumeracija `RUNNING`, `SUCCEES`, `FAILURE`, a koje se koriste za označavanje stanja čvora stabla. To je važno zato da implementacija stabla ponašanja bude lakše spojiva s aplikacijom uređivača stabla čija će implementacija biti opisana u nastavku.




*Slika 9 - Prikaz primjene oblikovnog obrasca Okvirna metoda na razred LeafNode*

## 2.3. Ispitivanje stabla ponašanja

Nakon izrade programskog ostvarenja stabla ponašanja bilo je potrebno ispitati i provjeriti ispravnost stabla. To se je ostvarilo pomiču ispitnih projekta koji se mogu pronaći unutar Visual Studia pri stvaranju novog projekta, točnije *Native Unit Test Project*[1]. Ta vrsta projekta služi za testiranje prirodnog (engl. *native*) C++ koda, tj. nepotpomognutog (engl. *unmanaged*). Nakon što se stvori novi ispitni projekt treba se dodati novi ispitni razred, odnosno *C++ Unit Test Class.* Za potrebe testiranja stabla ponašanja stvoreni su odgovarajući ispitni razredi kojima se ispituje njegova točnost. Primjer jednog takvog ispitnog slučaja kojim se ispituje funkcionalnost razreda `SelectorNode` može se vidjeti na sljedećem odsječku koda (Slika 10). Može se primijetiti da se ispitni razred definira s ključnom riječi `TEST_CLASS`, a ispitna metoda s ključnom riječi `TEST_METHOD`. Unutar jednog ispitnog razreda može biti neograničen broj ispitnih metoda različitog naziva. Ispitne metode se mogu pokrenuti unutar prozora *Test Explorer.*

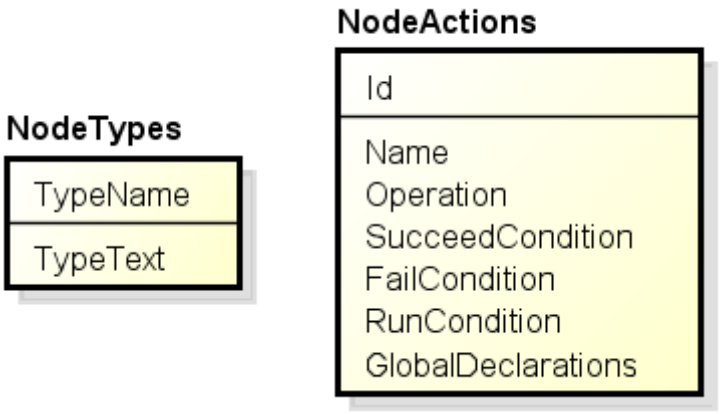
```
#include "stdafx.h"
#include "CppUnitTest.h"
#include "leaf_test_classes.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;
using namespace ai::behavior_tree;
using namespace ai;

TEST_CLASS(SelectorNodeTests) {
public:

	TEST_METHOD(SelectorNodeTest1) {
		auto leaf1 = new ActionFailNode();
		auto leaf2 = new ActionSuccNode();
		auto leaf3 = new ActionFailNode();
		auto seq = new SelectorNode();
		seq->addChildNode({ leaf1, leaf2, leaf3 });

		IBehavior* tree = new BehaviorTree(*seq);

		tree->tick();

		Assert::AreEqual(leaf1->counter, 1);
		Assert::AreEqual(leaf2->counter, 1);
		Assert::AreEqual(leaf3->counter, 0);
	}
}
```

*Slika 10 - Prikaz koda za ispitni slučaj nad razredom SelectorNode*

[1] Više informacija na - https://msdn.microsoft.com/en-us/library/hh270864.aspx

## 2.4. Izrada uređivača stabla ponašanja

Za potrebe stvaranja stabla potrebno je izraditi aplikaciju uređivača (engl. *editor*). Za to je odabran jezik C# i okruženju .NET Framework, odnosno Windows forme (engl. *Windows Forms*). Za potrebu izrade same aplikacije bilo je potrebno i stvoriti jednostavnu bazu podataka koja se sastoji od dviju tablica: `NodeTypes` i `NodeActions` (Slika 11). Kako se za prikaz stabla unutar aplikacije koristi .NET-ov razred `TreeView`, a koji koristi razred `TreeNode` za reprezentaciju čvora, iz razreda `TreeNode` se izvodi apstraktni razred `BehaviorTreeNode` iz kojeg se izvode svi ostali podtipovi čvorova stabla ponašanja. Tablica `NodeTypes` zato sadrži imena svih izvedenih konkretnih razreda čvorova stabla ponašanja kako bi se pomoću njih moglo dinamički stvarati instance određenih podtipova čvorova (engl. *Reflection*[2]). Dio koda koji prikazuje takav način stvaranja objekata prikazan je na slici 12 (Slika 12). Tablica `NodeActions` sadrži opis akcija stabla ponašanja na način koji je objašnjen u prethodnom poglavlju. Akcije se drže u bazi podataka kako bi se već napisane akcije mogle primijeniti na više različitih stabla ponašanja.




*Slika 11 - Prikaz tablica baze podataka*

[2] Više informacija na - https://msdn.microsoft.com/en-us/library/ms173183.aspx i https://msdn.microsoft.com/en-us/library/wccyzw83%28v=vs.110%29.aspx

```
private void nodeTypeComboBox_SelectionChangeCommitted(object sender, EventArgs e) {

_newNode = (BehaviorTreeNode) Activator.
              CreateInstance(Type.GetType(nodeTypeComboBox.SelectedValue.ToString()));

nodeSettingsFlowLayoutPanel.Controls.Clear();
nodeSettingsFlowLayoutPanel.Controls.AddRange(_newNode.SettingsControls);

}
```

*Slika 12 – Prikaz dijela koda koji stvara čvor na temelju imena razreda koji je enkapsuliran unutar comboBox-a, a koji se puni podatcima iz baze podataka.*

### 2.4.1. Obrasci uporabe

Umjesto da se opisuje sam tekst programa, u ovom poglavlju će biti fokus na obrascima uporabe (engl. Use Cases) koji sačinjavaju aplikaciju jer sama implementacija nije previše zanimljiva dok je način uporabe puno važniji (Slika 13).

UC1 – Preimenovanje stabla

- **Glavni sudionik**: Korisnik
- **Cilj:** Promjena imena stabla ponašanja
- **Sudionici:** Korisnik
- **Rezultat:** Korisnik je uspješno primijenio ime stabla
- **Željeni scenariji:**
    1. Korisnik unosi novo ime.
    2. Novo ime se prihvaća bez upozorenja da je nevaljano.
    3. Korisnika se obavještava da je ime uredu.
- **Alternativni scenariji:**
    1. Korisnik unosi novo ime.
    2. Novo ime se ne prihvaća zato što je nevaljalog formata (ima praznine) ili ne sadrži niti jedan znak.
    3. Korisnika se upozorava da je novo ime nevaljano.

UC2 – Spremanje datoteke kao

- **Glavni sudionik**: Korisnik
- **Cilj:** Spremiti stablo ponašanja tj. datoteku koja ga opisuje na specificiranu lokaciju na računalu
- **Sudionici:** Korisnik
- **Rezultat:** Korisnik je uspješno spremio datoteku
- **Željeni scenariji:**
    1. Korisnik odabire opciju za spremanje datoteke na specifičnu lokaciju (engl. *Save As*).
    2. Otvara mu se prozor gdje ga se traži da specificira lokaciju gdje želi spremiti datoteku i pod kojim imenom.
    3. Korisnik unosi potrebne podatke i pritišće gumb spremi.
    4. Datoteka se uspješno sprema.
- **Alternativni scenariji:**
    1. Nakon što korisnik specificira tražene podatke i pritisne gumb spremi aplikacija ga upozorava da datoteka s istim imenom na istoj lokaciji već postoji te pita korisnika dali želi odustati ili nastaviti.
    2. Korisnik nastavlja operaciju i sprema novu datoteku preko stare.

UC3 – Spremanje datoteke

- **Glavni sudionik**: Korisnik
- **Cilj:** Spremiti stablo ponašanja tj. datoteku koja ga opisuje na zadnju specificiranu lokaciju
- **Sudionici:** Korisnik
- **Rezultat:** Korisnik je uspješno spremio datoteku
- **Željeni scenariji:**
    1. Korisnik odabire opciju za spremanje datoteke (engl. *Save*).
    2. Datoteka se uspješno sprema.
- **Alternativni scenariji:**
    1. Ukoliko lokacija za spremanje nije nikada specificirana korisniku nudi mogućnost spremanja opisana s UC1, tj. opcija Spremi Kao.

UC4 – Otvaranje datoteke

- **Glavni sudionik**: Korisnik

- **Cilj:** Otvoriti datoteku koja se nalazi na računalu, a koja opisuje stablo ponašanja.
- **Sudionici:** Korisnik
- **Rezultat:** Korisnik je uspješno otvorio datoteku
- **Željeni scenariji:**
    1. Korisnik odabire opciju za otvaranje datoteke (engl. *Open*).
    2. Otvara se prozor gdje se korisnika traži da specificira lokaciju i ime datoteke koju želi otvoriti.
    3. Korisnik uspješno otvara datoteku.
- **Alternativni scenariji:**
    1. Ukoliko trenutna radna datoteka nije spremljena korisnika se prilikom pritiska na opciju za otvaranje datoteke pita dali želi spremiti trenutnu radnu datoteku prije nego li otvoriti drugu.
    2. Ako korisnik odabere da želi, preusmjerava se na prozor za spremanje (UC2), a ako ne želi operacija otvaranja se normalno nastavlja.

UC5 – Stvaranje nove datoteke

- **Glavni sudionik**: Korisnik
- **Cilj:** Stvoriti novu radnu datoteku unutar aplikacije
- **Sudionici:** Korisnik
- **Rezultat:** Korisnik je uspješno stvorio i otvorio novu radnu datoteku
- **Željeni scenariji:**
    1. Korisnik odabire opciju za stvaranje nove datoteke (engl. *New)*.
    2. Korisnik uspješno stvara i otvara novu datoteku.
- **Alternativni scenariji:**
    1. Ukoliko trenutna radna datoteka nije spremljena korisnika se prilikom pritiska na opciju za stvaranje nove datoteke pita dali želi spremiti trenutnu radnu datoteku prije nego li stvori i otvoriti novu.
    2. Ako korisnik odabere da želi, preusmjerava se na prozor za spremanje (UC2), a ako ne želi operacija otvaranja nove se normalno nastavlja.

UC6 – Generiranje teksta programa

- **Glavni sudionik**: Korisnik

- **Cilj:** Generirati kod za određeni jezik na temelju trenutne radne datoteke
- **Sudionici:** Korisnik
- **Rezultat:** Korisnik je uspješno generirao kod.
- **Preduvjet:** Korisnik je izgradio valjano stablo ponašanja (npr. svi listovi moraju biti akcije)
- **Željeni scenariji:**
    1. Korisnik odabire opciju za generiranje koda (engl. *Generate Code).*
    2. Korisniku se otvara prozor gdje ga se traži da odabere lokaciju gdje će se generirani kod spremiti (UC7), ime datoteke generiranog koda (UC8) i programski jezik generiranog koda (UC9).
    3. Korisnik uspješno odabire sve opcije i uspješno završava generiranje koda
- **Alternativni scenariji:**
    1. Ukoliko preduvjeti nisu zadovoljeni aplikacija odbija nastavak generiranja koda.
    2. Ako u koraku 2 nije odabrana lokacija ili odabrana ime nije valjano (sadrži razmake) tada korisnik ne može uspješno završiti generiranje koda prije nego li ispravi pogreške.

UC7 – Odabir lokacije datoteke

- **Glavni sudionik**: Korisnik
- **Cilj:** Odabrati valjanu lokaciju za datoteku
- **Sudionici:** Korisnik
- **Rezultat:** Odabrana je valjana lokacija datoteke
- **Željeni scenariji:**
    1. Korisnik odabire opciju za pregled i traženje lokacija na računalu (engl. *browse*).
    2. Korisnik odabire valjanu lokaciju na računalu.
- **Alternativni scenariji:**

UC8 – Odabir naziva datoteke

- **Glavni sudionik**: Korisnik
- **Cilj:** Odabrati valjan naziv izlazne datoteke

- **Sudionici:** Korisnik
- **Rezultat:** Odabrano je valjan naziv datoteke
- **Željeni scenariji:**
    1. Korisnik upisuje naziv datoteke
    2. Aplikacija javlja da je naziv valjan.
- **Alternativni scenariji:**
    1. Ukoliko uneseni naziv datoteke sadrži razmake ili je prazan aplikacija javlja da naziv nije valjan

UC9 – Odabir programskog jezika

- **Glavni sudionik**: Korisnik
- **Cilj:** Odabrati programski jezik za koji će se generirati kod.
- **Sudionici:** Korisnik
- **Rezultat:** Odabrana je programski jezik
- **Željeni scenariji:**
    1. Korisnik sa padajuće liste odabire jedan programski jezik
- **Alternativni scenariji:**

UC10 – Odabir željene akcije

- **Glavni sudionik**: Korisnik
- **Cilj:** Odabrati željenu akciju s liste akcija
- **Sudionici:** Korisnik
- **Rezultat:** Akcija je uspješno odabrana
- **Željeni scenariji:**
    1. Korisnik na listi akcija pronalazi željenu i odabire ju pritiskom.
    2. Željena akcija je uspješno odabrana.
- **Alternativni scenariji:**
    1. Na listi akcija se ne nalazi željena akcija zato što ne postoji ili zato što je potrebno osvježiti prikaz akcija (UC11).

UC11 – Osvježavanje prikaza akcija

- **Glavni sudionik**: Korisnik

- **Cilj:** Osvježiti prikaz akcija kako bi se prikazale sve akcije koje postoje u bazi podataka
- **Sudionici:** Korisnik i baza podataka
- **Rezultat:** Osvježavanje se je uspješno obavilo
- **Željeni scenariji:**
    1. Korisnik pritišće tipku za osvježivanje prikaza akcija (engl. *refresh*).
    2. Iz baze se dohvaćaju sve akcije i prikazuju se.
- **Alternativni scenariji:**

UC12 – Brisanje akcije

- **Glavni sudionik**: Korisnik
- **Cilj:** Obrisati akciju odabranu akciju s prikaza liste akcija i iz baze podataka
- **Sudionici:** Korisnik i baza podataka
- **Rezultat:** Odabrana akcija je uspješno obrisana
- **Preduvjet:** Odabrana je neka akcija
- **Željeni scenariji:**
    1. Korisnik odabire opciju za brisanje akcije (engl. *remove*).
    2. Akcija se uspješno briše s liste i iz baze podataka.
- **Alternativni scenariji:**

UC13 – Izmjena akcije

- **Glavni sudionik**: Korisnik
- **Cilj:** Izmijeniti odabranu akciju i osvježavanje baze podataka
- **Sudionici:** Korisnik i baza podataka
- **Rezultat:** Odabrana akcija je uspješno izmijenjena i baza podataka je izmijenjena
- **Preduvjet:** Odabrana je neka akcija
- **Željeni scenariji:**
    1. Korisnik izmjenjuje podatke o akciji.
    2. Korisnik prihvaća izmjene i baza podataka se osvježava.
- **Alternativni scenariji:**
    1. Korisnik izmjenjuje podatke o akciji.
    2. Korisnik odbacuje promjenjuje.

UC14 – Dodavanje akcije

- **Glavni sudionik**: Korisnik
- **Cilj:** Dodati novu akciju
- **Sudionici:** Korisnik i baza podataka
- **Rezultat:** Uspješno je dodana nova akcija
- **Željeni scenariji:**
    1. Korisnik odabire opciju za dodavanje nove akcije (engl. *add*)
    2. Korisniku se otvara novi prozor za unos imena nove akcije.
    3. Nakon što unese ispravno ime akcija se dodaje u bazu podataka i korisniku se omogućuje izmjena novonastale akcije (UC13)
- **Alternativni scenariji:**
    1. Ukoliko uneseno ime akacije nije valjano (ili ima praznina ili već postoji akcija s istim imenom) odbija se stvaranje nove akcije.

UC15 – Odabir čvora

- **Glavni sudionik**: Korisnik
- **Cilj:** Odabrati čvor stabla iz prikaza svih čvorova stabla
- **Sudionici:** Korisnik
- **Rezultat:** Uspješno je odabran neki čvor
- **Željeni scenariji:**
    1. Korisnik iz prikaza stabla pritišće na željeni čvor.
    2. Čvor se uspješno odabire.
- **Alternativni scenariji:**

UC16 – Dodavanje čvora

- **Glavni sudionik**: Korisnik
- **Cilj:** Dodati čvor u stablo.
- **Sudionici:** Korisnik
- **Rezultat:** Uspješno dodati čvor u stablo
- **Preduvjeti:**
    1. Pritisnut je desni klik prilikom odabiranja čvora roditelja
    2. Odabrani čvor nije list

3. Ako je odabrani čvor dekorater ne smije imati niti jedno dijete

- **Željeni scenariji:**
    1. Korisnik odabire opciju za dodavanje čvora (engl. *add*).
    2. Korisniku se otvara prozor za odabir tipa čvora i izmjenu postavki o njemu (UC17).
    3. Nakon odabira čvor se uspješno dodaje.
- **Alternativni scenariji:**

UC17 – Izmjena čvora

- **Glavni sudionik**: Korisnik
- **Cilj:** Izmijeniti odabrani čvor u stabla.
- **Sudionici:** Korisnik
- **Rezultat:** Uspješno izmijeniti odabrani čvor stabla
- **Preduvjeti:**
    1. Pritisnut je desni klik prilikom odabiranja čvora
- **Željeni scenariji:**
    1. Korisnik odabire opciju za izmjenu čvora (engl. *edit*).
    2. Korisniku se otvara prozor za odabir tipa čvora.
    3. Korisnik izmjenjuje postavke čvora.
    4. Nakon odabira čvor se uspješno izmjenjuje.
- **Alternativni scenariji:**
    1. Ako je čvor prije izmjene bio kompozit, a nakon izmjene postao dekorater ili list, korisnika se upozorava da bi se mogli obrisati neki čvorovi djece.
    2. Ako je čvor prije izmjene bio dekorater, a nakon izmjene postao list, korisnika se upozorava da bi se mogli obrisati neki čvorovi djece.

UC18 – Brisanje čvora

- **Glavni sudionik**: Korisnik
- **Cilj:** Obrisati odabrani čvor iz stabla
- **Sudionici:** Korisnik
- **Rezultat:** Uspješno je obrisan odabrani čvor iz stabla
- **Preduvjeti:**

1. Pritisnut je desni klik prilikom odabiranja čvora
2. Odabrani čvor nije korijen

- **Željeni scenariji:**
    1. Korisnik odabire opciju za brisanje čvora (engl. *edit*).
    2. Čvor se uspješno briše.
- **Alternativni scenariji:**

UC19 – Pomicanje čvora naprijed

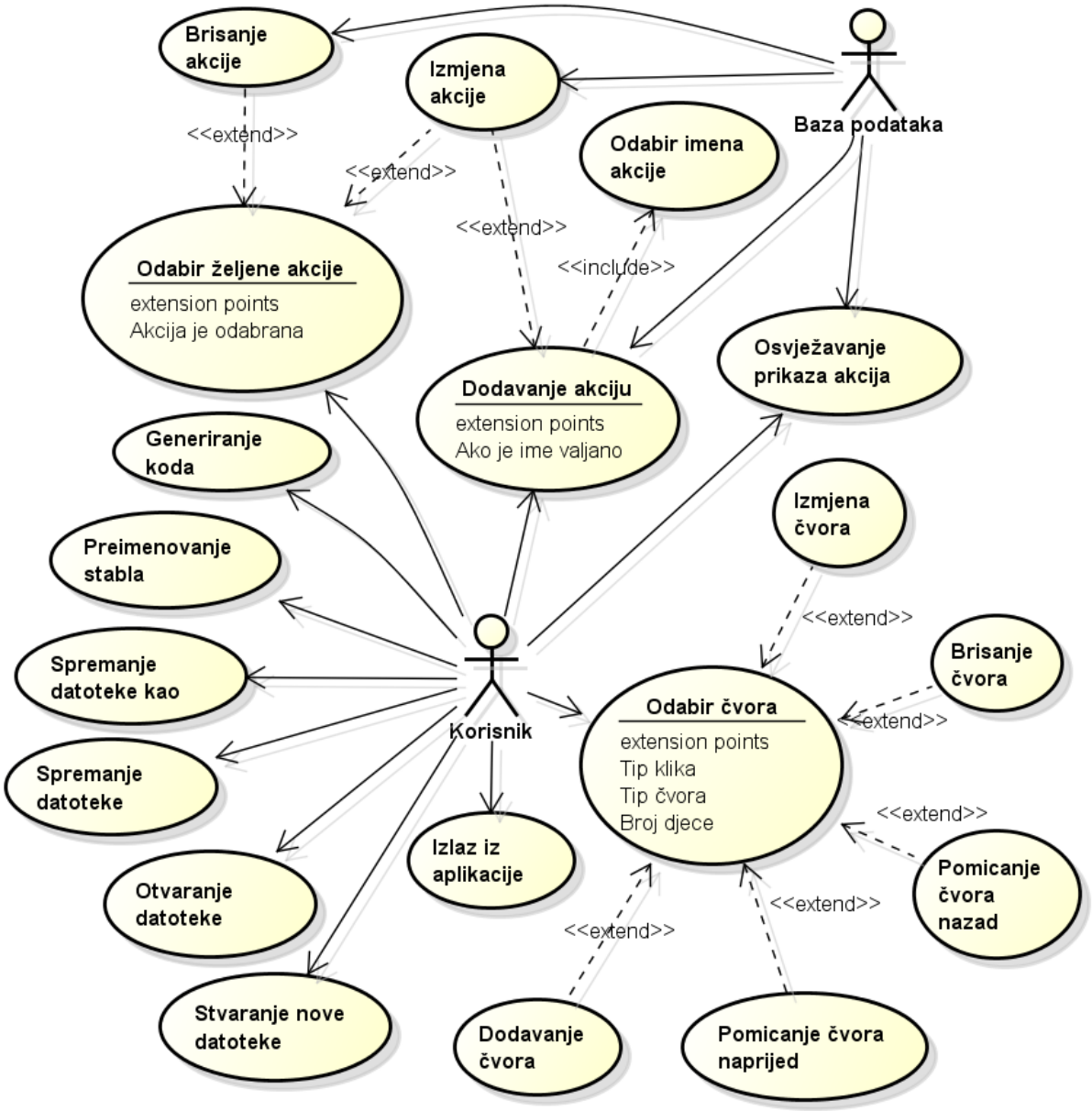
- **Glavni sudionik**: Korisnik
- **Cilj:** Pomaknuti odabrani čvor naprijed unutar roditelja
- **Sudionici:** Korisnik
- **Rezultat:** Odabrani čvor je uspješno pomaknut naprijed
- **Preduvjeti:**
    1. Pritisnut je desni klik prilikom odabiranja čvora roditelja
    2. Odabrani čvor ima „brata“ ispred sebe.
- **Željeni scenariji:**
    1. Korisnik odabire opciju za pomicanje čvora naprijed čvora (engl. *up*).
    2. Odabrani čvor se uspješno pomiče naprijed.
- **Alternativni scenariji:**

UC20 – Pomicanje čvora nazad

- **Glavni sudionik**: Korisnik
- **Cilj:** Pomaknuti odabrani čvor nazad unutar roditelja
- **Sudionici:** Korisnik
- **Rezultat:** Odabrani čvor je uspješno pomaknut nazad
- **Preduvjeti:**
    1. Pritisnut je desni klik prilikom odabiranja čvora roditelja
    2. Odabrani čvor ima „brata“ iza sebe.
- **Željeni scenariji:**
    1. Korisnik odabire opciju za pomicanje čvora nazad čvora (engl. *down*).
    2. Odabrani čvor se uspješno pomiče nazad.
- **Alternativni scenariji:**

UC21 – Izlaz iz aplikacije

- **Glavni sudionik**: Korisnik
- **Cilj:** Izaći iz aplikacije
- **Sudionici:** Korisnik
- **Rezultat:** Aplikacija je uspješno ugašena
- **Željeni scenariji:**
    1. Korisnik odabire opciju za gašenje aplikacije (engl. *exit*).
    2. Aplikacija se uspješno gasi.
- **Alternativni scenariji:**
    1. Ako radna datoteka nije spremljena korisnika se upozorava ako želi spremiti datoteku prije gašenja programa.



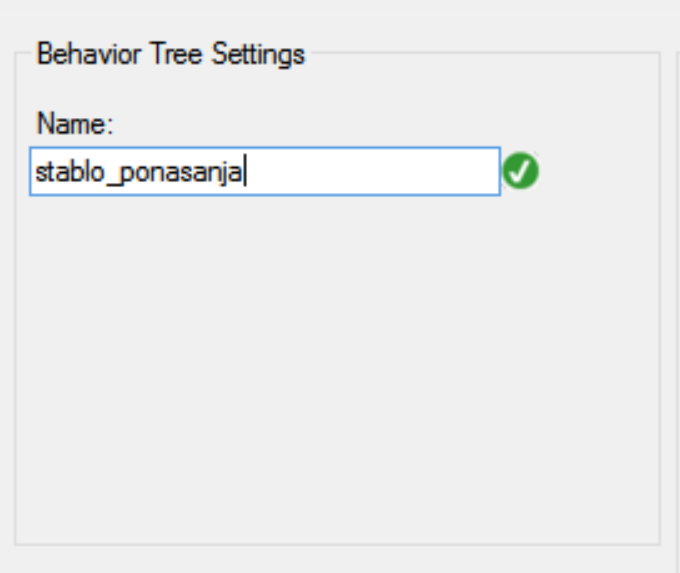

*Slika 13 – Prikaz dijagrama obrazaca uporabe*

# 3. Opis procesa izrade stabla ponašanja

U ovom dijelu će se prikazati proces izrade stabla ponašanja uz pomoć aplikacije uređivača.

## 3.1. Odabir imena stabla ponašanja

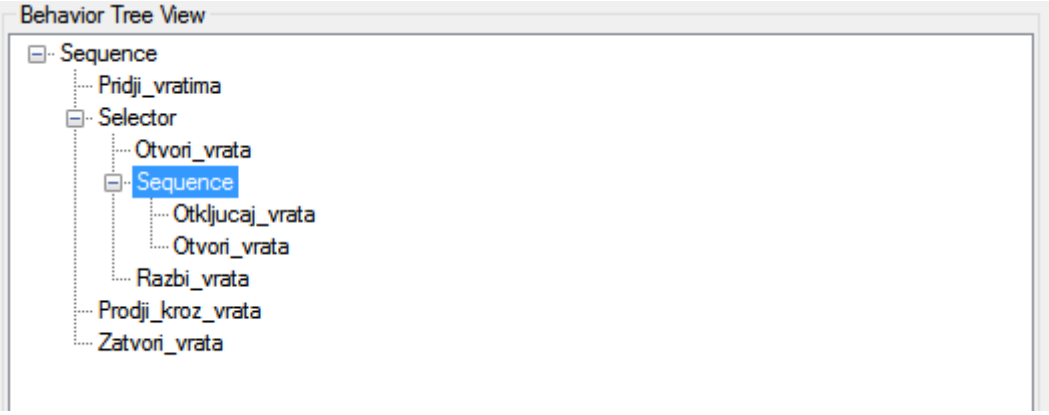
Na početku stvaranja stabla korisnik treba odabrati ime stabla ponašanja koje će se koristiti u generiranom kodu, a predstavljat će baš to stablo ponašanja. Ime stabla mora zadovoljavati uvjet da nema praznina. Sljedeća slika (Slika 14) prikazuje jedan valjan unos imena stabla ponašanja.




*Slika 14*

## 3.2. Stvaranje stabla

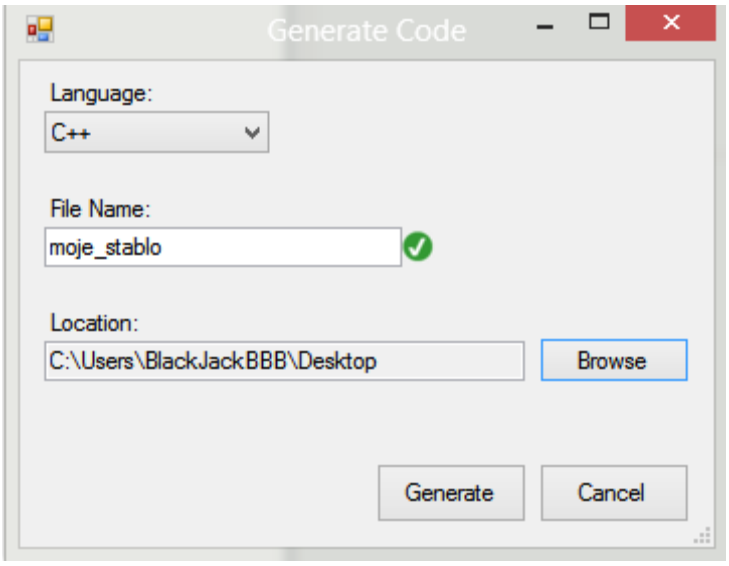
Nakon odabira imena stabla korisnik treba stablo pomoću ponuđenih operacija izgraditi stablo. Sljedeća slika (Slika 15) prikazuje stablo koje sa slike 2 (Slika 2). Uvjeti da stablo bude ispravno je da su svi listovi akcije, tj. da ne bude listova koji nisu akcije.




*Slika 15*

## 3.3. Generiranje koda stablo ponašanja

Nakon što je korisnik izradio željeno stablo treba odabrati opciju *File->Generate Code* kako bi mu se ponudila otvorio prozor za specifikaciju generiranje u kojemu treba odabrati programski jezik za koji želi generirati kod, naziv izlazne datoteke i lokaciju na računalu gdje će se generirane datoteke nalaziti (Slika 16). Naziv izlazne datoteke ne smije sadržavati praznine.




*Slika 16*

## 3.4. Prikaz generiranog stabla ponašanja za C++

Konačno treba prikazati i produkt aplikacije uređivača, a to je generirani C++ kod koji koristi stablo ponašanja čija je implementacija opisana u prethodnim poglavljima. Sljedeće dvije slike (Slika 17 i Slika 18) prikazuju izlazne datoteke za C++. Važno je napomenuti da su kodovi za akcije iz prethodnog primjera prazni zbog jednostavnosti prikaza no u stvarnosti to će biti popunjeno s odgovarajućim kodom.

```
#include <leaf_node.h>
#include <selector_node.h>
#include <sequence_node.h>
#include <inverter_node.h>
#include <repeater_node.h>
#include <succeeder_node.h>
#include <until_fail_node.h>
namespace stablo_ponasanja {
	class Pridji_vratima : public ai::behavior_tree::LeafNode {
	private:
		// Global declarations

		bool runCondition() override;
		bool succeedCondition() override;
		bool failCondition() override;
		void operation() override;
	};
	class Prodji_kroz_vrata : public ai::behavior_tree::LeafNode {
	private:
		// Global declarations

		bool runCondition() override;
		bool succeedCondition() override;
		bool failCondition() override;
		void operation() override;
	};
	class Zatvori_vrata : public ai::behavior_tree::LeafNode {
	private:
		// Global declarations

		bool runCondition() override;
		bool succeedCondition() override;
		bool failCondition() override;
		void operation() override;
	};
	class Razbi_vrata : public ai::behavior_tree::LeafNode {
	private:
		// Global declarations

		bool runCondition() override;
		bool succeedCondition() override;
		bool failCondition() override;
		void operation() override;
	};
	class Otkljucaj_vrata : public ai::behavior_tree::LeafNode {
	private:
		// Global declarations

		bool runCondition() override;
		bool succeedCondition() override;
		bool failCondition() override;
		void operation() override;
	};
	class Otvori_vrata : public ai::behavior_tree::LeafNode {
	private:
		// Global declarations
		bool runCondition() override;
		bool succeedCondition() override;
		bool failCondition() override;
		void operation() override;
	};
	ai::behavior_tree::BehaviorTree* generate();
}
```

*Slika 17 – Prikaz datoteke moje_stablo.h za stablo iz prethodnog primjera*

```
#include "moje_stablo.h"

using namespace behavior_tree_name;
using namespace ai::behavior_tree;

BehaviorTree* generate() {
	auto node0 = new SequenceNode();
	node0->addChildNode(new Pridji_vratima());
	auto node1 = new SelectorNode();
	node1->addChildNode(new Otvori_vrata());
	auto node2 = new SequenceNode();
	node2->addChildNode(new Otkljucaj_vrata());
	node2->addChildNode(new Otvori_vrata());
	node1->addChildNode(node2);
	node1->addChildNode(new Razbi_vrata());
	node0->addChildNode(node1);
	node0->addChildNode(new Prodji_kroz_vrata());
	node0->addChildNode(new Zatvori_vrata());

	return new BehaviorTree(*node0);
}

bool Pridji_vratima::runCondition() {}
bool Pridji_vratima::succeedCondition() {}
bool Pridji_vratima::failCondition() {}
bool Pridji_vratima::operation() {}
bool Prodji_kroz_vrata::runCondition() {}
bool Prodji_kroz_vrata::succeedCondition() {}
bool Prodji_kroz_vrata::failCondition() {}
bool Prodji_kroz_vrata::operation() {}
bool Zatvori_vrata::runCondition() {}
bool Zatvori_vrata::succeedCondition() {}
bool Zatvori_vrata::failCondition() {}
bool Zatvori_vrata::operation() {}
bool Otvori_vrata::runCondition() {}
bool Otvori_vrata::succeedCondition() {}
bool Otvori_vrata::failCondition() {}
bool Otvori_vrata::operation() {}
bool Razbi_vrata::runCondition() {}
bool Razbi_vrata::succeedCondition() {}
bool Razbi_vrata::failCondition() {}
bool Razbi_vrata::operation() {}
bool Otkljucaj_vrata::runCondition() {}
bool Otkljucaj_vrata::succeedCondition() {}
bool Otkljucaj_vrata::failCondition() {}
bool Otkljucaj_vrata::operation() {}
```

*Slika 18 – Prikaz datoteke moje_stablo.cpp za stablo iz prethodnog primjera*

# 4. Mogućnost nadogradnje

## 4.1. Mogućnost nadogradnje stabla ponašanja

Stablo ponašanja izrađeno u ovome radu može biti nadograđeno na razne načine. Od toga da se dodaju novi tipovi čvorova (npr. *Paralel*) do toga da se pri stvaranja stabla rade optimizacije poput više roditeljstva (engl. *multi parenting*) koje služi za eliminaciju ponavljanja istih podstabala tako da omogućuje da jedan čvor djeteta može imati više roditelja (tada stablo zapravo postaje usmjereni aciklički graf (engl. *directed acyclic graph*)) te se tako štedi memorijski prozor. Također, stablo ponašanja se može stvarati i poboljšavati uz pomoću evolucijskih algoritama[3] (npr. genetski algoritam GA). Osim ovih postoje i drugi načini funkcionalne nadogradnje stabla ponašanja, ali oni neće biti razmatrani u ovom završnom radu.

## 4.2. Mogućnost nadogradnje uređivača

Mogućnost nadogradnje aplikacije uređivača ovisi i o implementaciji samog algoritma stabla ponašanja zato što uređivač generira kod koji stvara stablo ponašanja. No neovisno o tome sama aplikacija uređivača se može nadograditi na više načina kako bi njena funkcionalnost bila veća. Prva stvar koja bi se trebala i mogla implementirati u aplikaciji je mogućnost da se korisnik u svakome trenutku može vratiti potez unazad (engl. *undo*) ili unaprijed (engl. *redo*) čime bi se korisniku omogućilo da bude efikasniji u radu i da si može priuštiti grešku prilikom stvaranja stabla. Druga funkcionalnost koja bi se mogla dodati u aplikaciju je ta da se kod za stvaranje stabla može generirati i za druge programske jezike (Java, C#, JavaScript, Python itd.), a ne samo za C++, no preduvjet za tu nadogradnju je ta da „negdje“ mora postojati implementacija algoritma stabla ponašanja za te jezike. Sljedeće što bi se moglo dodati aplikaciji je to da se za pisanje operacija akcija unutar aplikacije uvede posebni skriptni jezik, a koji bi se zatim prevodio u odgovarajuće izlazne jezike. Također bi se mogao ugraditi i program za pronalaženje pogrešaka (engl. *debugger*) unutar stvorenog stabla. Ovo su samo neke od mogućih dodatnih funkcionalnosti, ali mogućnosti za nadogradnju su neograničene.

[3] D. Perez, M. Nicolau, M. O'Neill, A. Brbazon. Evolving Behavior Tree for Mario AI Competition Using Grammatical Evolution, *http://ncra.ucd.ie/papers/66240123.pdf*, 3.6.2015.

# Zaključak

Temu ovog završnom rada motivirana je informacijom da se za ostvarivanje kompleksnih inteligencija umjesto konačnih automata koriste stabla ponašanja. Na tu informaciju sam naišao prilikom razvijanja igre Pac-Man, a gdje sam umjetnu inteligenciju ostvario oblikovnim obrascem Stanje (engl. *State Design Pattern*), tj. objektno orijentiranom inačicom konačnih automata. Iz navedenog razloga odlučio sam da ću u ovome radu proučiti kako stablo ponašanja funkcionira i kako se izrađuje. Također, prilikom pretraživanja interneta „naletio“ sam na informaciju da razni sustavi za izradu igara (engl. *game engine*) poput *Unity-a*, *Unreal Engina* i *CryEngina* sadrže svoju implementaciju stabla ponašanja i da sadrže uređivač za izradu stabla, stoga sam odlučio da ću izraditi svoj uređivač stabla kojim bi se izgradnja stabla bitno pojednostavila. Uređivači iz navedenih sustava sadrže sve dodatke koji su navedeni u poglavlju 2.7., a koji bi se mogli ugraditi u uređivač iz ovoga rada. Konačno trebam reći da sam zadovoljan s onime što sam ostvario u ovome radu, no i da postoji mjesta za napredak i nadogradnju.

# Literatura

Champandard, Alex. 2012. *Understanding the Second Generation of Behavior Trees and Preparing for Challenges Beyond .* 23.. Veljača. https://www.youtube.com/watch?v=n4aREFb3SsU.

Pereira, Renato. 2014. *An Introduction to Behavior Trees.* 25.. Srpanj. http://guineashots.com/2014/07/25/an-introduction-to-behavior-trees-part-1/.

Simpson, Chris. 2014. *Behavior trees for AI: How they work.* 7.. Srpanj. http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php.

# Sažetak

Stablo ponašanja (engl. *Behavior Tree*) danas je jedan od najraširenijih načina prikaza umjetne inteligenciju u robotici i industriji video igara. Ovaj način prikaza umjetne inteligencije krase jednostavnost izrade i mogućnost nadogradnje i ponovne uporabe dijelova zbog čega danas često zamjenjuju konačne automate. U ovome radu se opisuje izrada stabla ponašanja pogonjenog na događaje. Rad se sastoji od dva dijela: izrade algoritma stabla ponašanja unutar jezika C++ i izrade uređivača stabla pomoću jezika C# i .NET Framework-a.

Nowadays the Behavior Tree is one of the most popular way to create artificial intelligence in robotics and video games. This way of showing artificial intelligence adorn the simplicity and the ability to upgrade and re-use of parts which is why today often replaces finite state machines. This work describes creation of event driven Behavior Tree. The work is combined of two parts: creation of algorithm in C++, that describes tree, and creation of editor in C# and .NET Framework that is used for creating and editing tree.

# Ključne riječi

stablo ponašanja; događaji; umjetna inteligencija; konačni automati; uređivač; C++

behavior tree; events; artificial intelligence; finite state machine; editor; C++