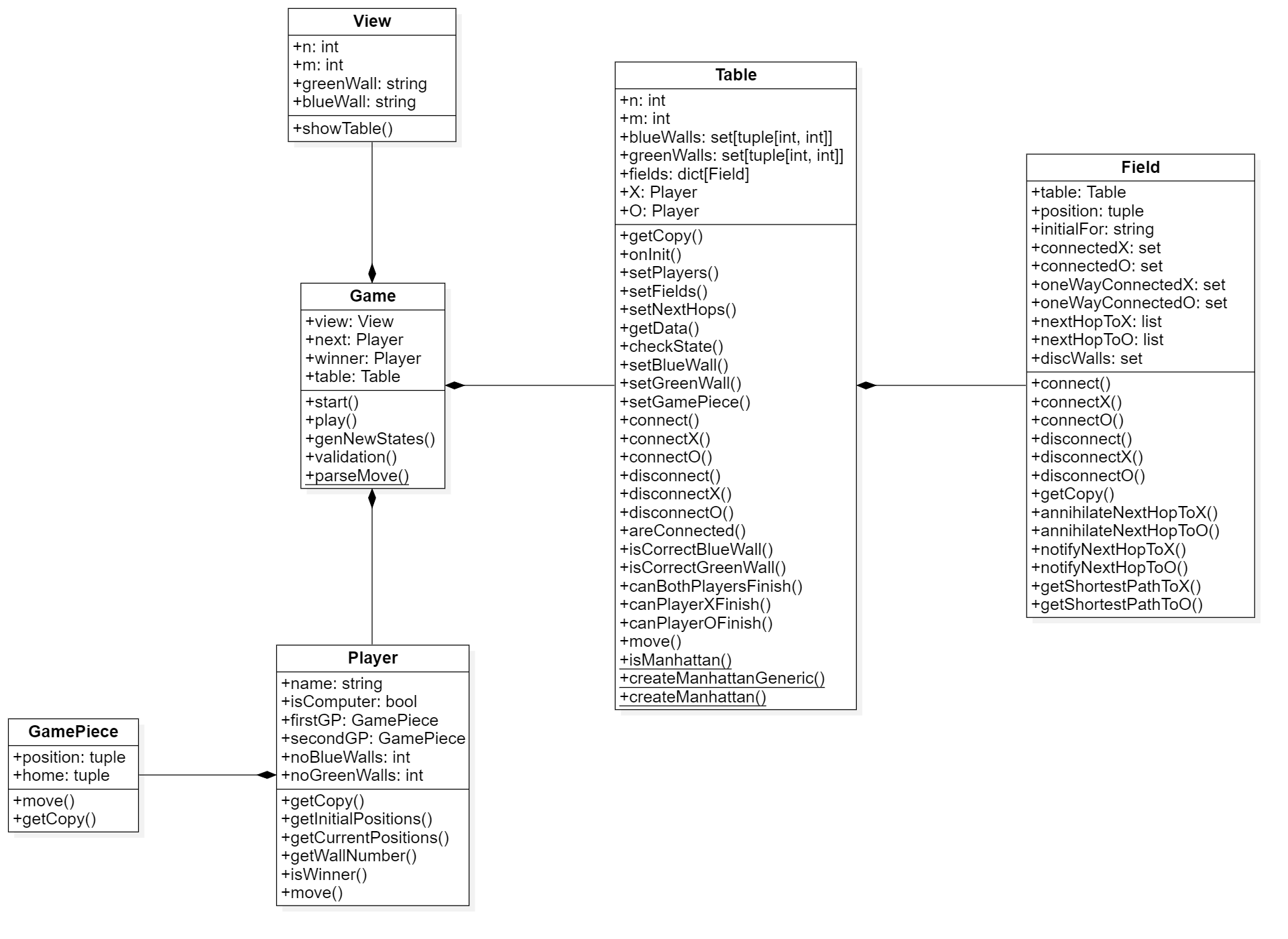
Теодора Коцић 17190

Стефан Алексић 16995

Извештај пројекта ,֦,Blockade”

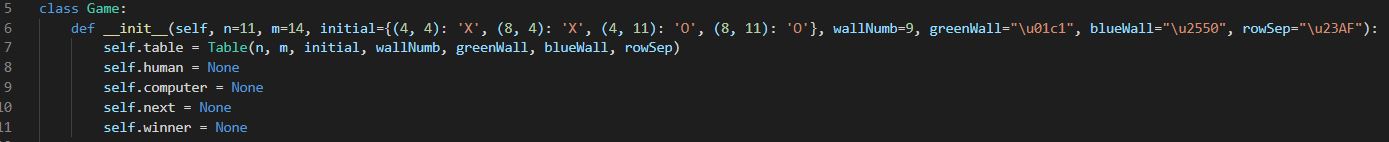
На слици 1 приказан је класни дијаграм у којем су наведене све до сада формиране класе у сврху решавања проблема.



*Слика 1 Класни дијаграм*

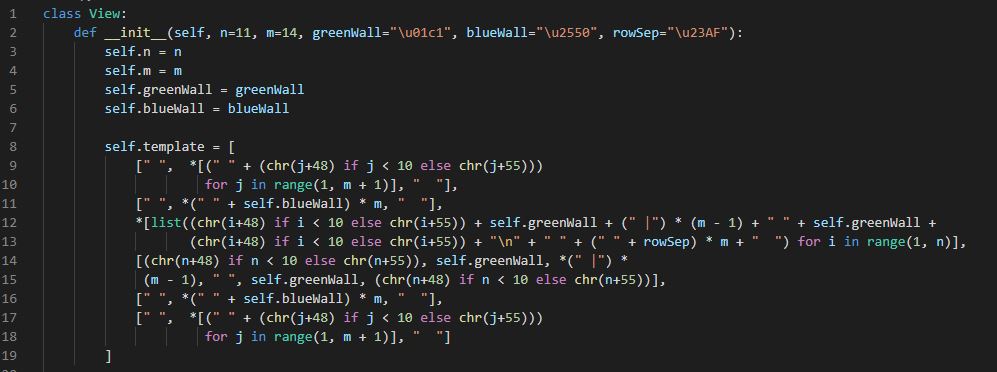
1. Фаза – Формулација проблема и интерфејс
2. Фаза – Оператори промене стања

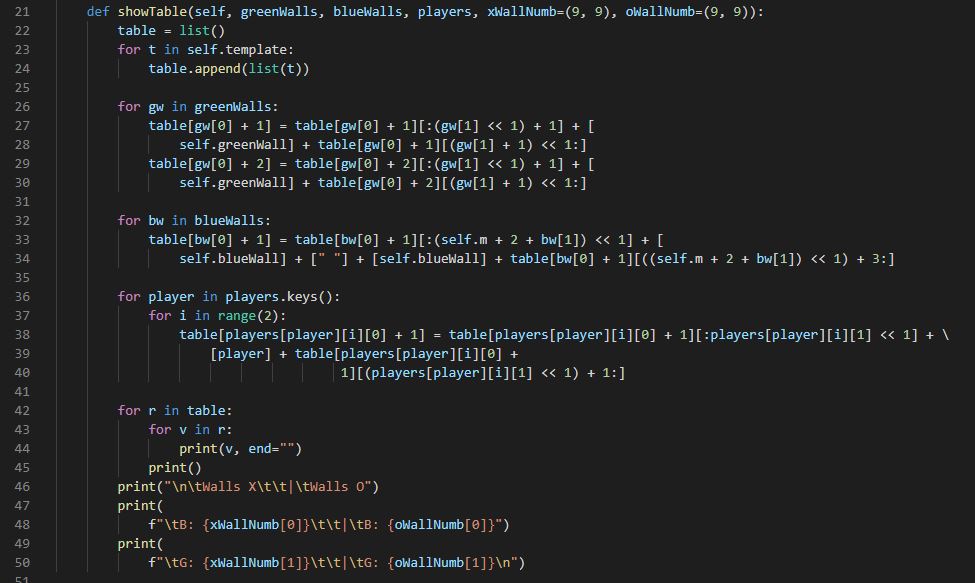
За потребе дефинисања стања проблема игре користи се класа *Game* која садржи инстанцу објекта класе *Table*, као и две инстанце класе *Player*.

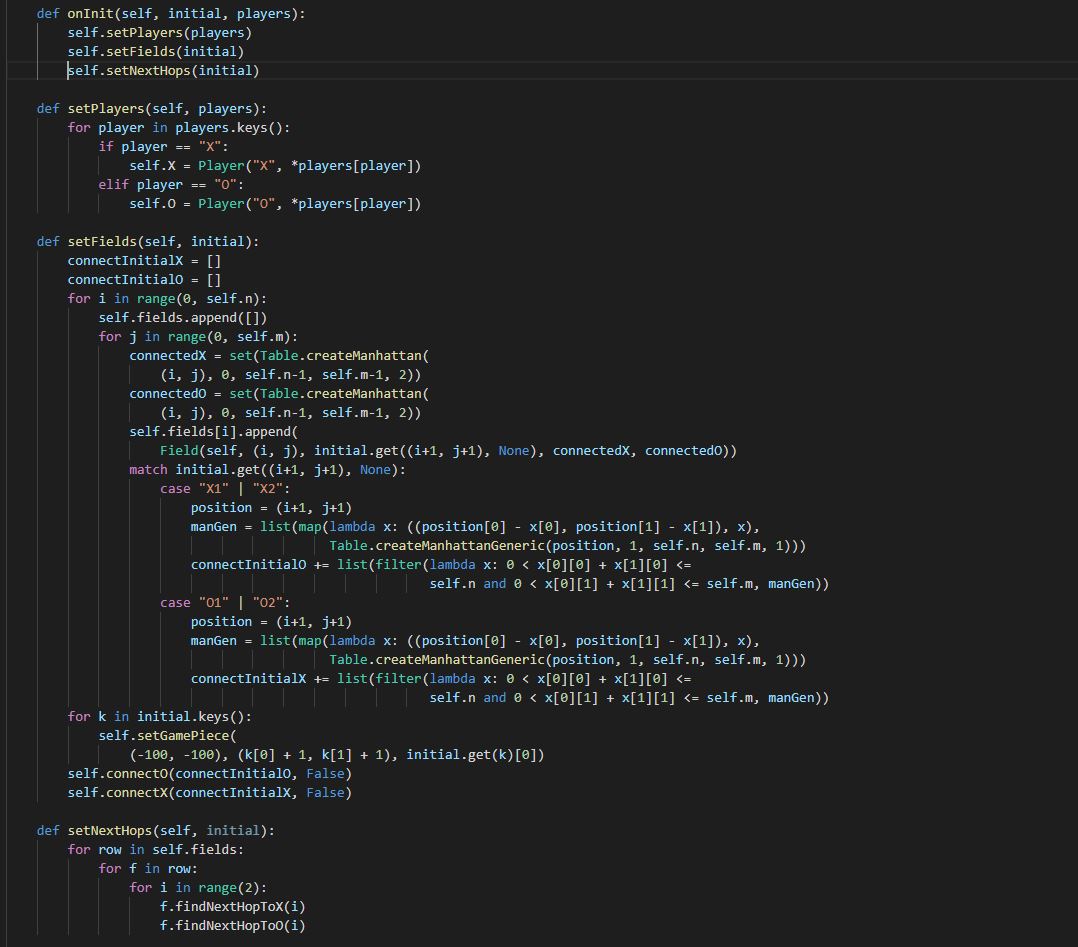
*Слика 2 Конструктор класе Game*

У оквиру класе *Table*, као што се може видети у 7. линији кода на слици 2, врши се иницијализација табле која се користи у игри. Параметрима *n* и *m* дефинишемо димензије табле, атрибут типа речник *initial* садржи вредност почетних позиција сваког од четири играча на табли. У променљивој *wallNumb* памти се информација о броју зидова једне боје коју поседује сваки од играча (дакле број зидова играча X, односно играча О је 2\**wallNumb*). Остали параметри конструктора служе за приказ табле у конзоли апликације.

Класу *View* користимо само у сврхе представљања изгледа читаве игре у конзоли. Поред конструктора у оквиру којег се врши исцртавање табле. Класа садржи и функцију *showTable(...)* којом се креира шаблон којим се приказује тренутно стање табле. Приликом покретања апликације поред табле у конзоли приказане су и све четири фигуре на својим почетним позицијама и то је одрађено у функцији *onInit(...)* класе *Table* и овај део кода приказан је на слици 5.



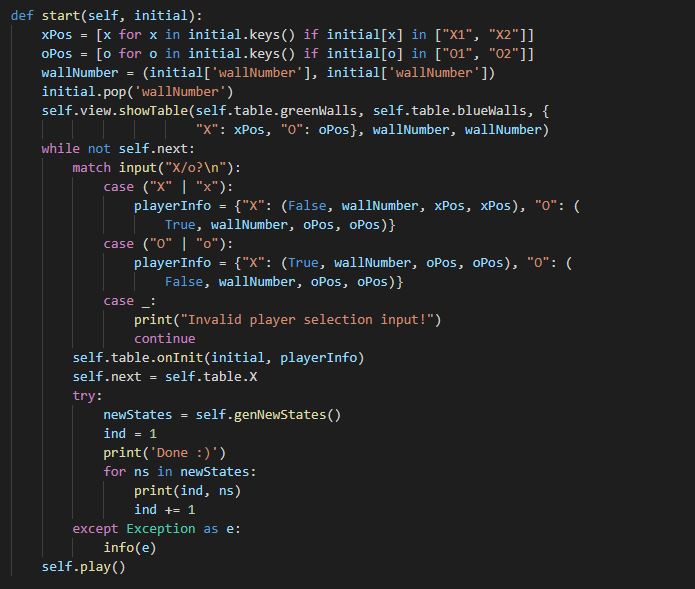


**

*Слика 3, 4, 5 Конструктор класе View  
 Функција ShowTable класе View*

*Постављање фигура на одговарајуће почетне позиције*

Унутар функције *onInit(...)* класе *Table* можемо видети још три позива функција којима је управо обезбеђено постављање почетног стања. Функцијом *setNextHop(...)* се користи у сврхе формирања путање за сваку од фигура приликом генерисања нових стања, тј. након сваког одиграног потеза.

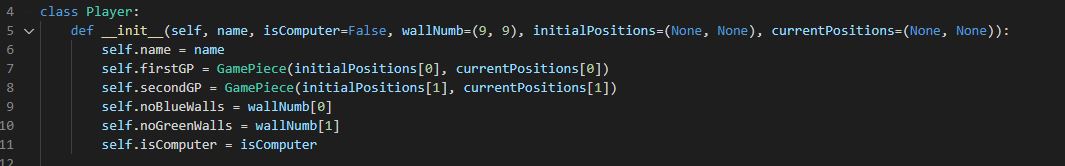


*Слика 6 Функција за постављање почетног стања игре*

За представљање почетног стања игре користи се функција *start*, чланица класе *Game*, приказана на слици 6.

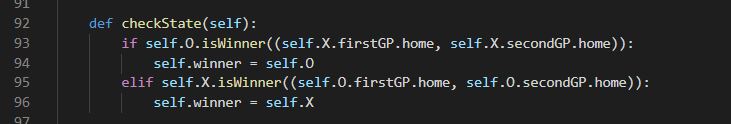
У функцији *start* постављају се вредности за почетне позиције играча, као и број зидова које играч поседује. Играч X увек ће бити први на потезу када игра отпочне, док се кориснику пружа могућност одабира фигуре са којом ће играти игру. Корисник уносом карактера “X”/”x” бира опцију да игра први, односно други уносом карактера “О”/”о”.

Конструктор класе *Player* за први аргумент има *name* којим се води рачуна о томе ко је први на потезу, док се другим аргументом поставља флег који води рачуна о томе да ли је играч човек или рачунар (због потреба израде каснијих фаза пројекта), затим се редом као аргументи прослеђују број зидова за сваку од две боје зидова које добијају играчи на почетку игре, иницијалне позиције обе фигуре датог играча, као и тренутне позиције играча. Такође класа *Player* садржи две инстанце класе *GamePiece* која служи за дефинисање једне фигуре и један од атрибута ове класе је атрибут *home* који чува податак о почетној позицији фигуре датог играча. Због тога се фигуре (две фигуре по играчу) креирају као нови објекти типа *GamePiece* којима се прослеђују иницијалне позиције које се памте у класи *Player* (7. и 8. линија кода на слици 7). Други атрибут *position* памти информацију о тренутној позицији фигуре на табли.



*Слика 7 Конструктор класе Player*

На крају сваког потеза врши се испитивање да ли је игра окончана датим потезом позивом функције *checkState()* чија се имплементација може видети на слици 8. Испитује се да ли је управо одиграним потезом играч стао на иницијално поље противника, чиме се прекида игра, јер је он уједно и победник. То се проверава кроз позив функције *isWinner(...)* чланице класе *Player*. У оквиру функције *isWinner(...)* испитује се да ли се тренутна позиција играча налази у одговарајућој листи позиција. У позиву наведене функције унутар функције *checkState(…)* као аргумент који представља описану листу прослеђују се иницијалне позиције играча О за играча X и обратно, позиције играча X за играча О.



*Слика 8 Провера да ли је игра окончана*



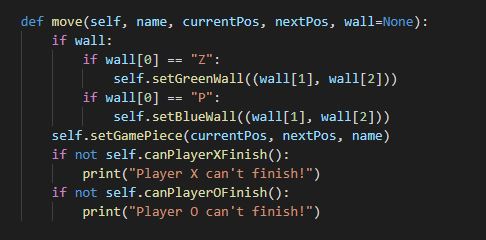
*Слика 9 Функција која обезбеђује приказ произвољног стања игре*

Функција којом се обезбеђује приказ произвољног стања игре је *play*, док функција *parseMove* служи да из уноса потеза са тастатуре очита вредности за одговарајуће параметре и уколико је дошло до некоректног уноса обавести корисника о томе, у супротном врати изглед самог потеза који је корисник одиграо.

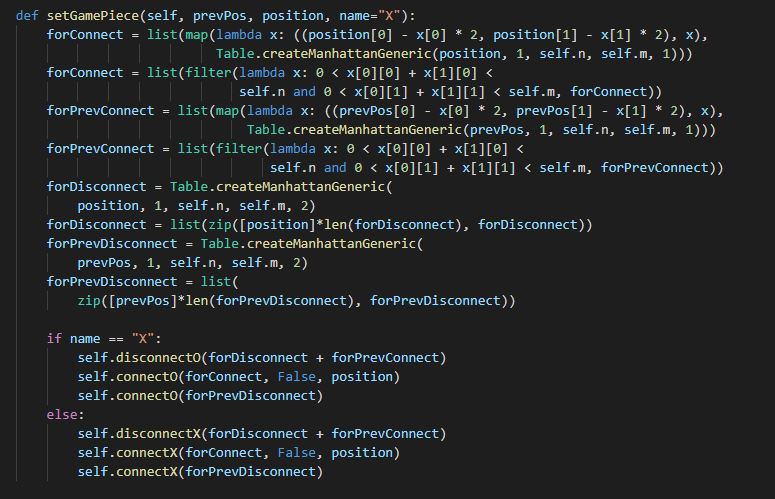
У функцији *play* позива се функција *move* из класе *Table* којом се постављају зидови уколико их играч поседује и врши се промена места саме фигуре, позивом функције *move* класе *Player*. Такође у функцији обезбеђује се и промена редоследа играња потеза, тј. уколико је на потезу последњи био играч X сада се поставља да је на потезу играч О и обратно.

Функција *move* класе *Table* позива oстале функције чланице ове класе: *setBlueWall(...), setGreenWall(...)* и *setGamePiece(...)*. На слици 10 приказана је имплементација функције *move,* функцијекојом се мења тренутно стање на табли након сваког потеза играча постављањем одговарајућег зида (функције *setBlueWall(...)* и *setGreenWall(...)*) на прослеђену позицију као и померањем жељене фигуре. Функцијом *setGamePiece(...)* врши се постављање конекција за дату фигуру у зависности од тренутне и наредне позиције на табли. Већи део кода из имплементације функције је релевантан за касније фазе израде пројекта, међутим оно што је овде важно јесте да се за прослеђену фигуру остварују конекције, тј. овиме се ограничава њено кретање по табли. Тако да фигура може да пређе на поља чије је растојање једнако два (уколико то поље није заузето), при чему се израчунавање врши по Manhattan Pattern-у. Manhattan Pattern израчунава растојање између два вектора без коришћења функција квадрирања и кореновања. Сваким позивом ове функције врши се и испитивање да ли је дошло до сценарија где је неко од иницијалних поља фигура ограђено и уколико јесте враћа се порука о томе.

У функцији приказаној на слици 11, такође се кроз конекције обезбеђује да уколико се на позицији која јесте правилно одигранa као нови потез, налази нека друга фигура, померање тренутне фигуре на суседну позицију (позиција која се налази између тренутне и жељене) буде могуће. Ова функција у суштини прави нове и укида неке од постојећих веза противнику због услова да играч може да се креће по хоризонтали и вертикали и за по једно поље у случају да је жељено поље заузето, међутим ово ће бити објашњено са више детаља касније у тексту.

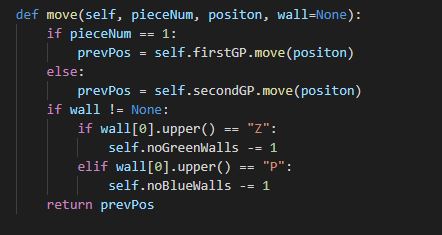


*Слика 10 Функција која садржи логику одигравања потеза на табли*

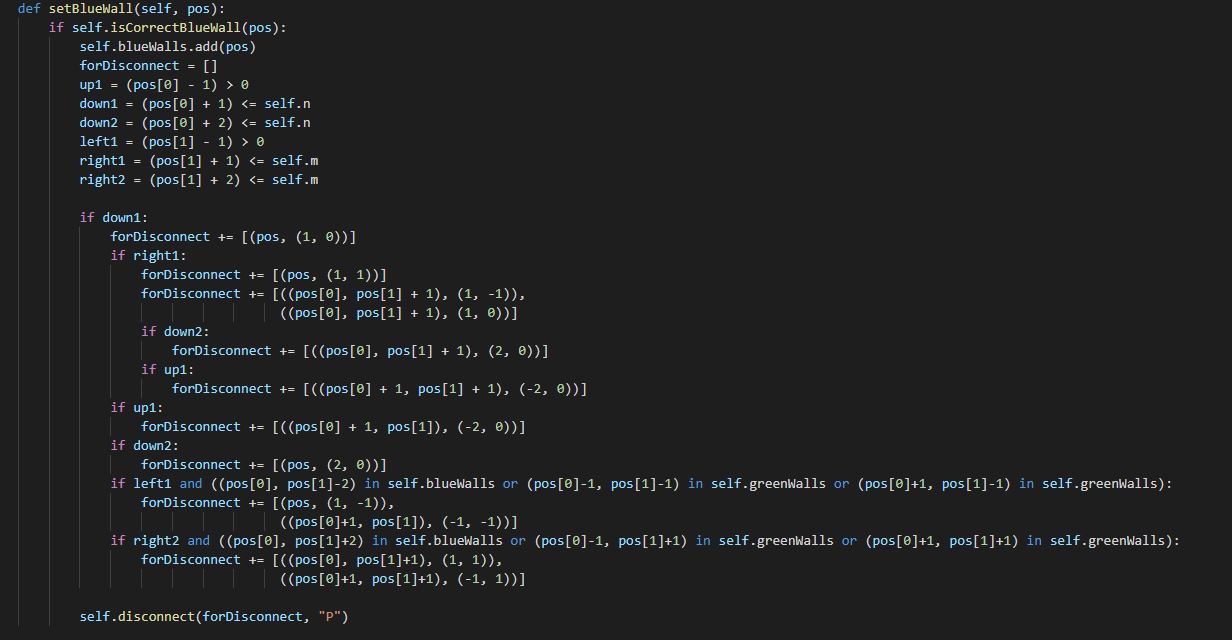


*Слика 11 Функција која ограничава кретање пешака*

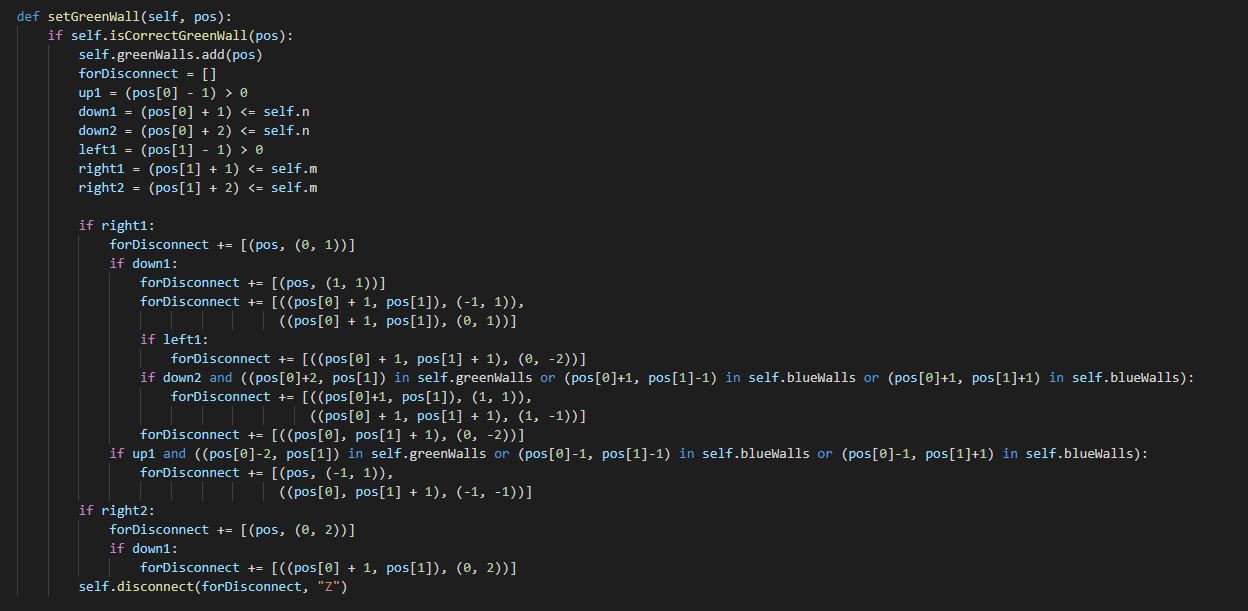
На слици 12 може се видети изглед функције *move* класе *Player.* Повратна вредност функције је стање у којем се фигура налазила пре одигравања новог потеза играча. Ако играч остане без зидова потез може да садржи само одабир фигуре и нову позицију фигуре, што се дефинише променљивом *wall*. У случају да играч још увек поседује зидове у зависности од прослеђене боје зида укупан број зидова те боје се декрементира за један.



*Слика 12 Функција којом се мења стање играча*

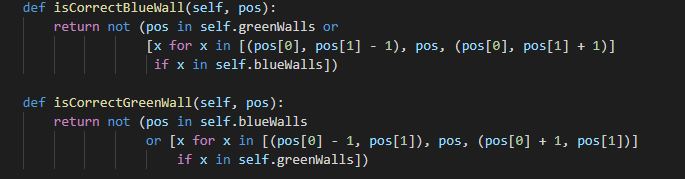


*Слика 14 Функција за постављање плавих зидова*



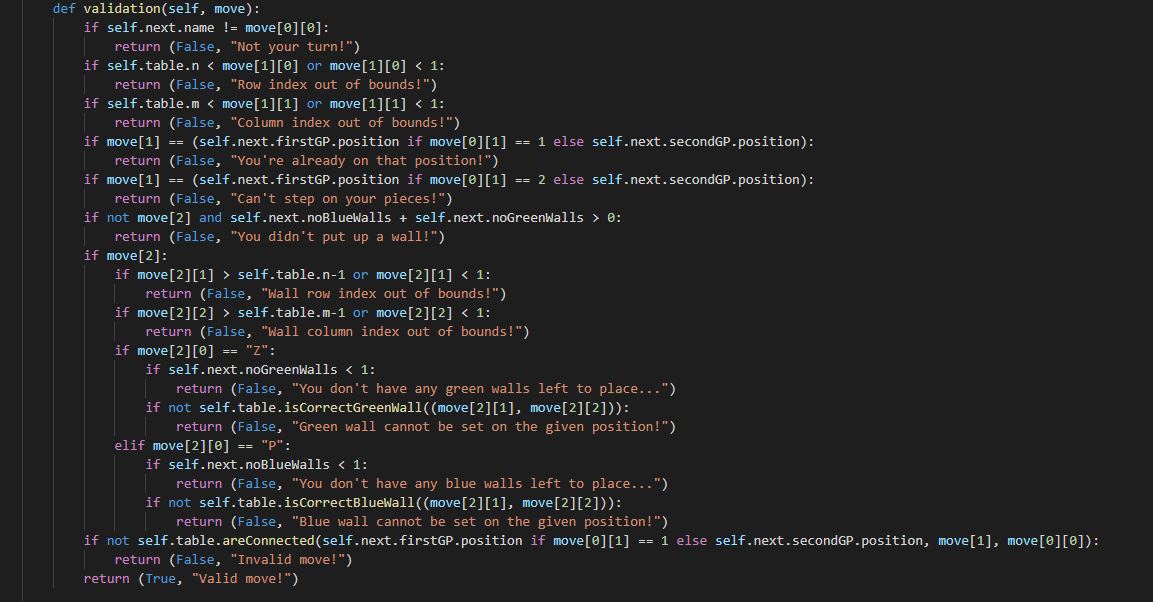
*Слика 15 Функција за постављање зелених зидова*

Уколико је валидна позиција на коју корисник жели да постави зид функцијама *setBlueWall(...)* и *setGreenWall(...)* је то могуће и одрадити. Испитивање валидности жељене позиције на коју би ишао зид врши се функцијама класе *Table*, приказаним на сликама 14 и 15. Такође постављањем зидова на одређене позиције смањује се број могућих потеза који следе након одигравања тренутног, јер није валидно преклапање зидова, или пролазак пешака кроз зидове. То се контролише позивом функције *disconnect(...)* након постављања зида на задату позицију.

**

*Слика 16 Функције за испитивање валидности нових позиција зида*

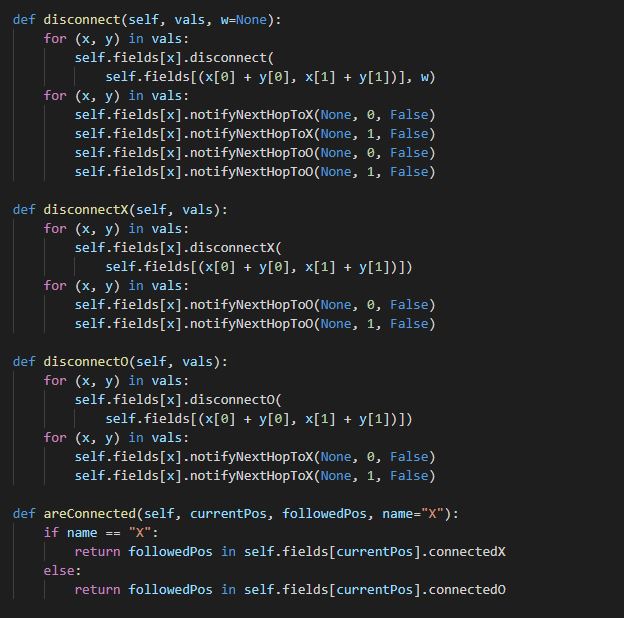
Дакле функција *isCorrectBlueWall* враћа вредност True уколико се прослеђенa позиција (скуп (врста, колона)) зидa не налази у листи позиција (листа садржи скупове(врста, колона) за сваку позицију на којој већ постоји зид на табли) постављених зидова зелене боје, као и плаве боје и уколико се позиција у жељеној врсти: (врста, колона + 1), односно (врста, колона - 1) не налази у листи постављених зидова плаве боје. Слично и за проверу постављања новог зеленог зида, разлика је у томе што се у овом случају испитује да ли позиције (врста -1, колона) и (врста + 1, колона) не припадају листи постављених зидова зелене боје.

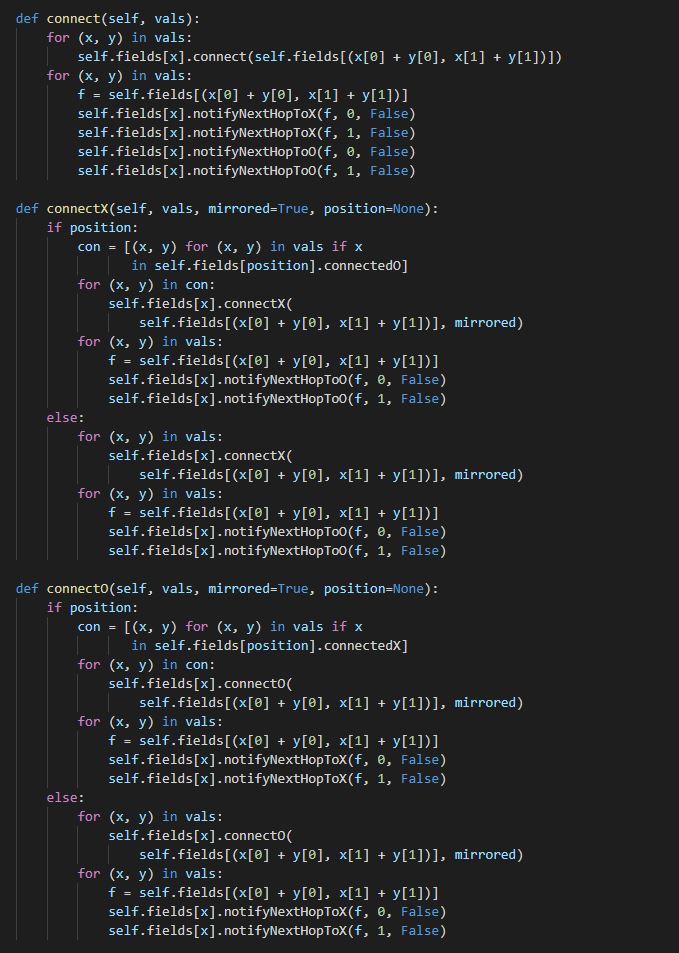


*Слика 17 Функција правилности потеза*

Такође, испитивање валидности сваког потеза (померај пешака, позиције зидова, унос потеза, …) одрађено је унутар функције *validation(...)* и то се може видети на слици 14. Уколико је корисник одиграо потез у којем је за индекс врсте, односно колоне поставио вредност која је већа од димензија табле, односно мања од јединице, апликација ће пријавити грешку. Грешка ће се јавити у следећим случајевима: корисник жели да одигра потез у којем за нову позицију наводи тренутну позицију своје фигуре, или наводи иницијалну позицију дате фигуре; играч још увек има зидове, а у потезу није навео боју и позицију зида коју жели да постави на табли; уколико играч у потезу наводи зид који жели да постави на одређену позицију, а зидова тражене боје више нема или је за позицију зида унео параметре који излазе из опсега (о овоме је раније било речи кроз објашњења дата за функције *isCorrectBlueWall(...)* и *isCorrectGreenWall(...)*) и на самом крају испитује се да ли део потеза корисника који се односи на померај фигуре прати правила кретања по табли (по два поља лево односно десно гледано за врсту табле, по два поља доле односно горе гледано за колону табле, по једно поље гледано по дијагонали у односу на тренутну позицију или по једно поље уколико је потез валидан, а жељено поље које представља нову позицију заузето).

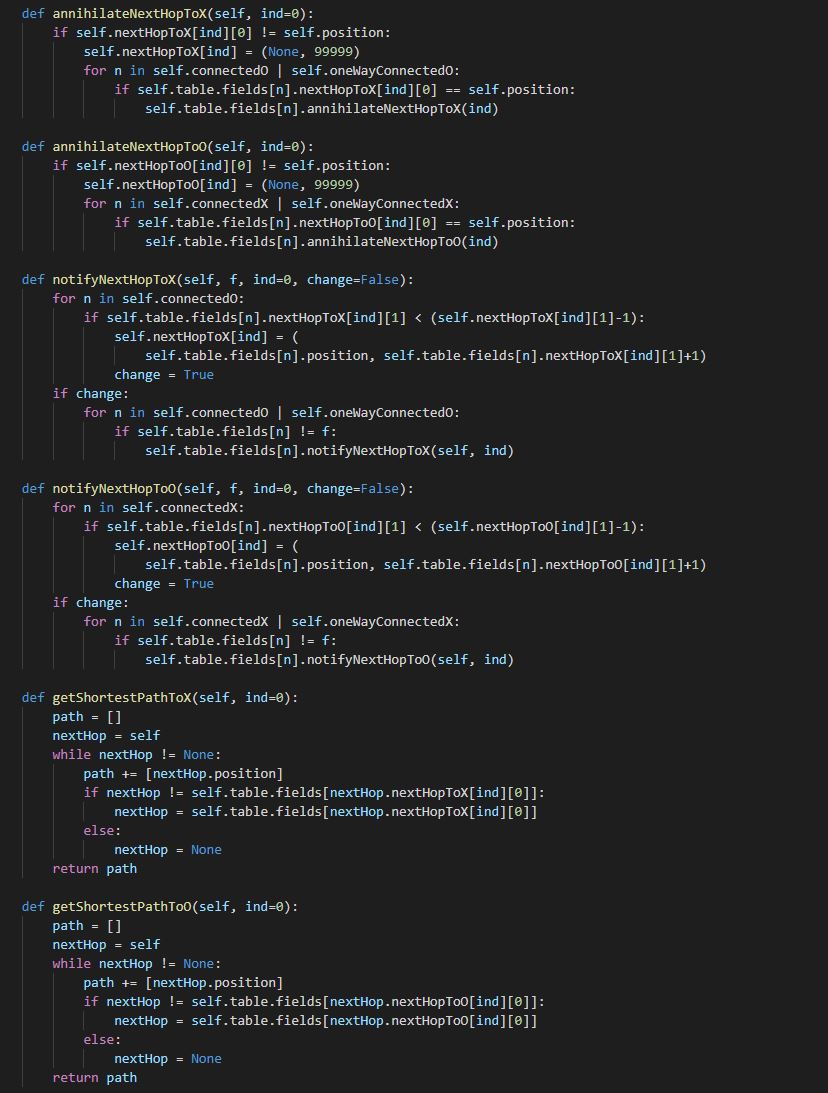
У класи *Field* постоје функције које касније користи класа *Table* и управо ове чланице класе *Table* представљају већи део логике која стоји иза сваког потеза: померања фигуре и/или постављања зидова. Функцијама *connect* i *disconnect* појединачних фигура омогућава се додавање/укидање веза за поједина поља у зависности од тога где је фигура померена и где је зид постављен. Уколико се постави зид на одређено поље, оно ће изгубити везу са околним пољима и тиме је спречено ,,прескакање“ зидова у оквиру игре, односно ако се фигура постави на одређено поље добија конекције са пољима која су по хоризонтали, односно вертикали на удаљености два, или удаљености 1 за случај кретања по дијагонали.





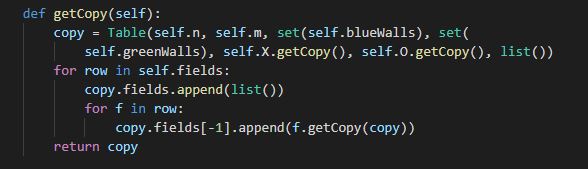
*Слика 18, 19 Функције за креирање и укидање веза међу пољима табле*

У споменутим функцијама класе *Field* врши се и ,,креирање“ путање за дату фигуру у зависности од стања након одиграног потеза, што се може видети у имплементацији функција *notifyNextHopToX(…)*, *notifyNextHopToO(…)*, *annihilateNextHopToX(…)*, *annihilateNextHopToO(…)*, *getShortestPathToX(…)* и *getShortestPathToO(…)*. Ове функције представљају алгоритам чија се имплементација базира на познатом алгоритму под називом *Distance Vector*.

**

*Слика 20 Функције којима је имплементирана модификована верзија Distance Vector алгоритма*

Функцијом *getCopy(...)* која је чланица класе *Table* на основу тренутног стања на табли и одиграног потеза креира се ново стање тако што се најпре направи копија тренутног стања табле након чега се над том копијом одиграва задати потез.



*Слика 21 Креирање новог стања на табли*

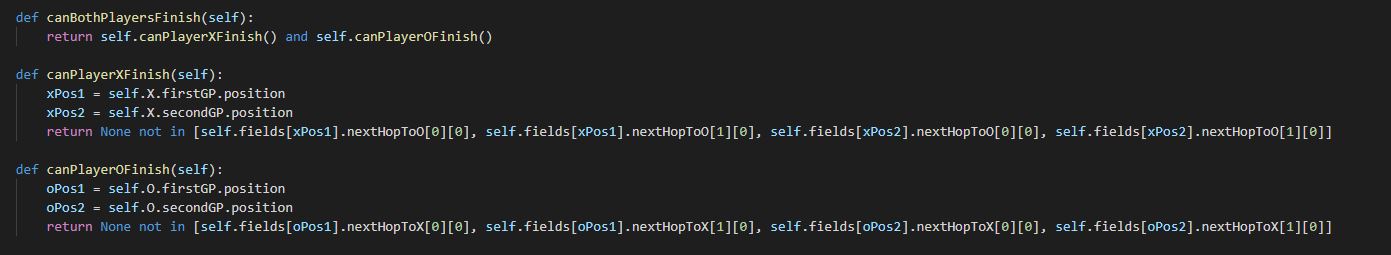
У функцији приказаној на слици 18 врше се позиви и функција *getCopy(...)* класа *Field* i *Player*, редом, чија је улога креирања новог стања табле након одиграног потеза. Имплементације ове две функције дате су на слици 19 и 20.



*Слика 22, 23 Функција getCopy класе Field*

*Функција getCopy класе Table*

Имплементација функције *genNewSates(...)* којом се врши генерисање свих могућих стања у зависности од тренутног стања на табли као и управо одиграног потеза приказана је на слици 21. У оквиру ове функцији постоји и позив функције *canBothPlayersFinish(...)* која на основу информација која добија од функција приказаним на слици 24.



*Слика 24 Функције којим асе врши провера да ли је неки од играча блокиран тако да не може да стигне до иницијалног поља противника*

Функција која проверава унос задатог потеза је функција *parseMove(...)* класе *Game,* док функција која позива ову функцију и захтева унос новог потеза уколико претходни позив врати грешку је функција *play(...)* класе *Game*. Обе ове функције приказане су на слици 9 раније у документу. Промена тренутног стања се добија као последица позива функције *move(...)* над објектом класе *Table,* док се то ново стање на табли може видети у конзоли због позива функције *showTable(...)* над објектом типа *View*. Описани позиви функција могу се видети у оквиру функције *play(...)* на слици 9. док се имплементација функција *move(...)* класе *Table* и функције *showTable(...)* класе *View* могу видети редом на сликама 10 и 4. За проверу краја игре користи се функција *checkState(...)* класе *Table* (слика 10)*,* која се такође позива у функцији *play(...)*, а у оквиру које се позива функција *isWinner(...)* класе *Player*, која врши проверу да ли је играч стао на неку од две иницијалне позиције противника (слика 25).



*Слика 25 Функција којом се проверава да ли је играч победио након одиграног потеза*