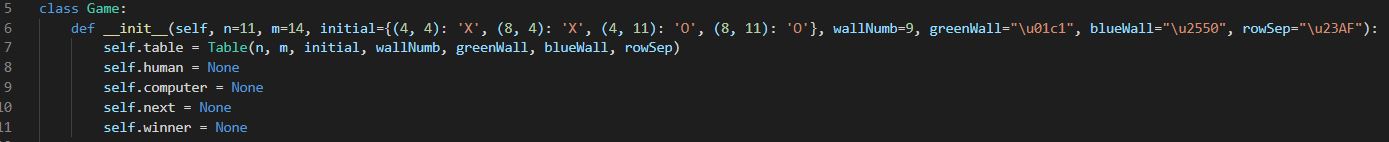
# Извештај пројекта ,֦,Blockade”

1. Фаза – Формулација проблема и интерфејс

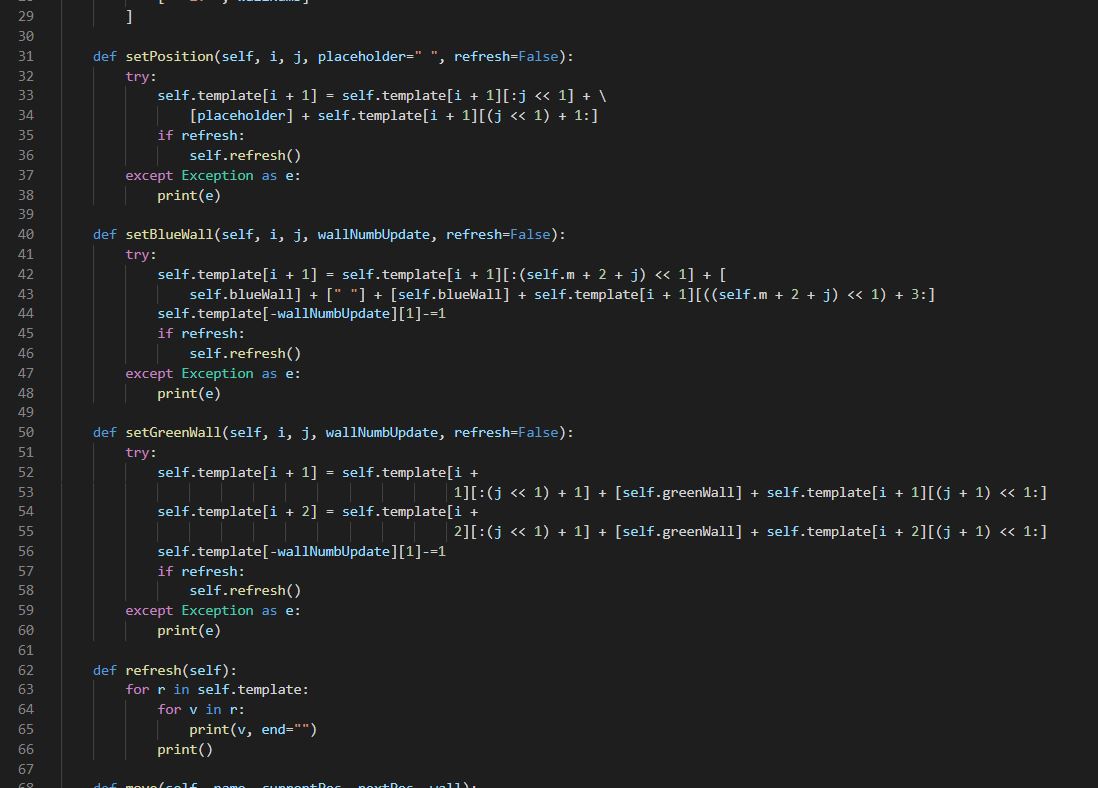
За потребе дефинисања стања проблема игре користи се класа *Game* која садржи инстанцу објекта класе *Table*, као и две инстанце класе *Player*.

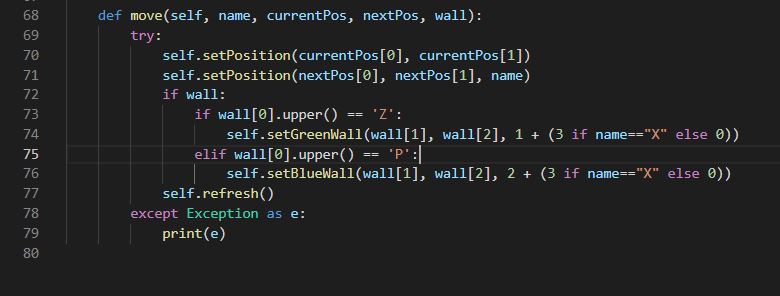
*Слика 1 Конструктор класе Game*

У оквиру класе *Table*, као што се може видети у 7. линији кода на слици 1, 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 као и играча који се налазе на својим почетним позицијама, класа садржи и функције којима се врши исцртавање тренутног изгледа табле након сваког извршеног потеза.



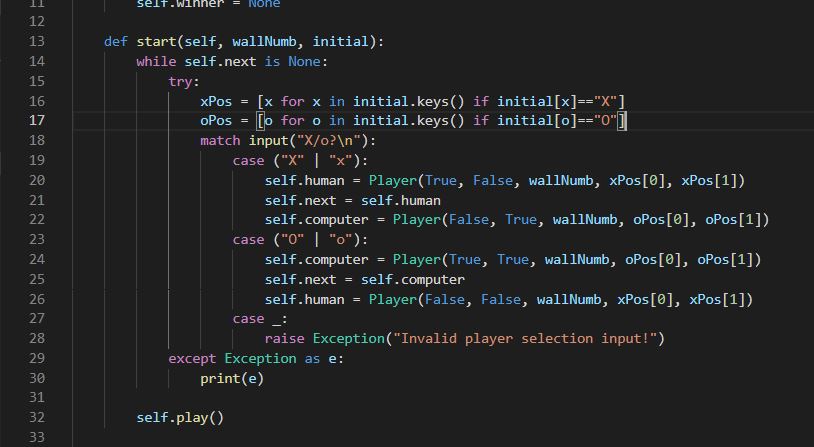




*Слика 2, 3, 4 Kонструктор класе View  
 Функције класе View*

Функцијом *setPosition(...)*, *setBlueWall(...)* и *setGreenWall(...)* врши се промена приказа тренутног стања табеле. Функцијом *refresh* освежава се приказ табле, док се позив функције *move* врши унутар класе *Table*, о којој ће убрзо бити више речи у тексту.

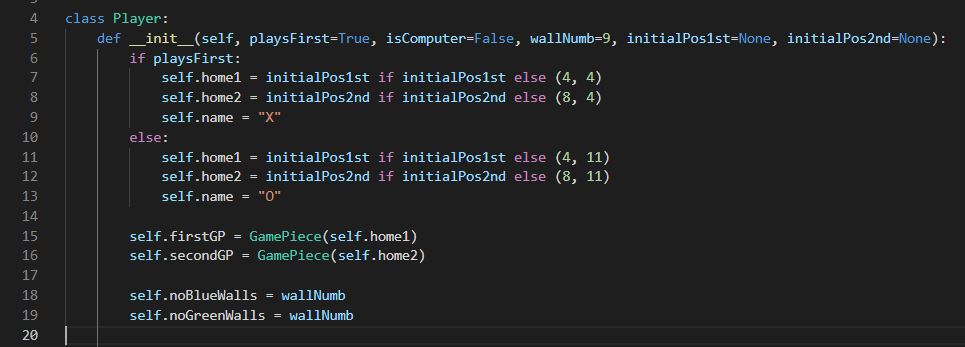
За представљање почетног стања игре користи се функција *start*, чланица класе *Game*, приказана на слици 5.



*Слика 5 Функција за постављање почетног стања игре*

У функцији *start* постављају се вредности за почетне позиције иргача, играч X увек ће бити први на потезу када игра отпочне, док се кориснику пружа могућност одабира да ли ће као играч X играти човек или рачунар (унос са тастатуре у 18. линија кода). Уколико корисник унесе “X”/”x” бира опцију да човек игра први, а рачунар други (први случај за *case*) или уносом “O”/”o” поставља сценарио тако да рачунар игра први, а човек други (други случај за *case*).

Конструктор класе *Player* као први аргумент прима логичку вредност којом 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, затим се редом као аргументи прослеђују број зидова за сваку од две боје зидова које добијају играчи на почетку игре и иницијалне позиције обе фигуре датог играча, као што се може видети на слици 6.

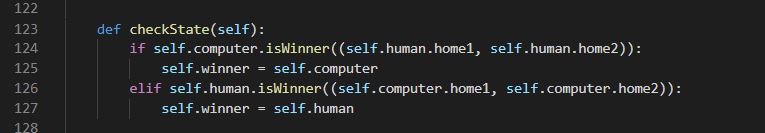


*Слика 6 Конструктор класе Player*

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 7 Функција која обезбеђује приказ произвољног стања игре*

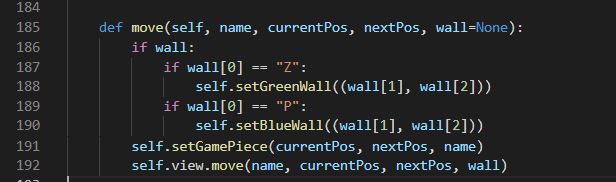


*Слика 8 Провера да ли је игра окончана*

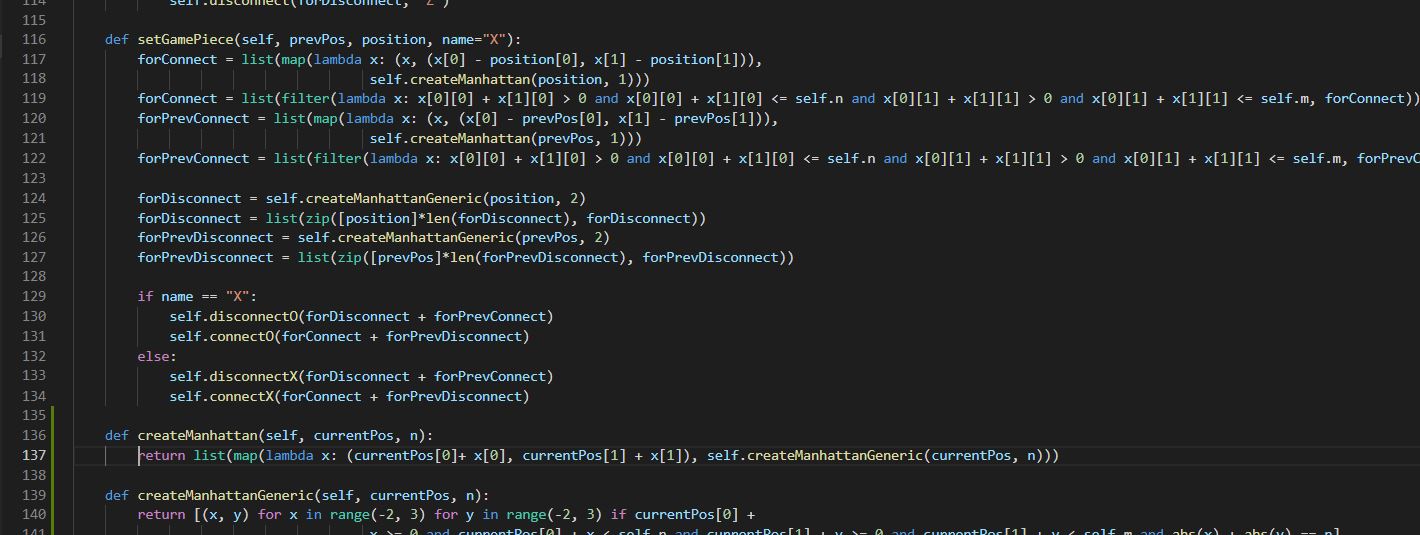
Функција којом се обезбеђује приказ произвољног стања игре је *play*, док функција *parseMove* служи да из уноса потеза са тастатуре очита вредности за одговарајуће параметре и уколико је дошло до некоректног уноса обавести корисника о томе, у супротном врати изглед самог потеза који је корисник одиграо.

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 9 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 2 (уколико то поље није заузето) по Manhattan Pattern-u (услов из дефиниције правила дате игре). Manhattan Pattern израчунава растојање између два вектора без коришћења функција квадрирања и кореновања. У функцији приказаној на слици 10, такође се кроз конекције обезбеђује да уколико се на позицији која јесте валиднo одигранa као нови потез, налази нека друга фигура, померање тренутне фигуре на суседну позицију (позиција која се налази између тренутне и жељене) буде могуће.

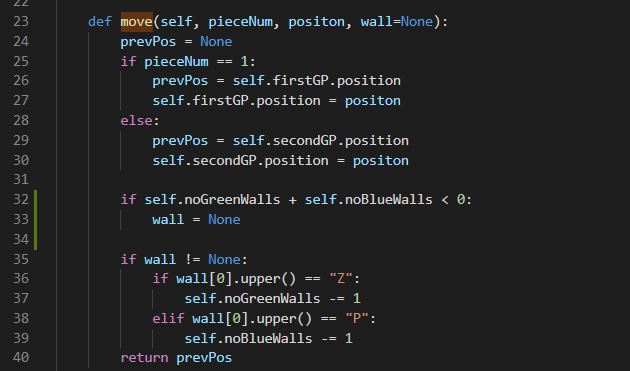


*Слика 9 Функција која садржи логику одигравања потеза на табли*

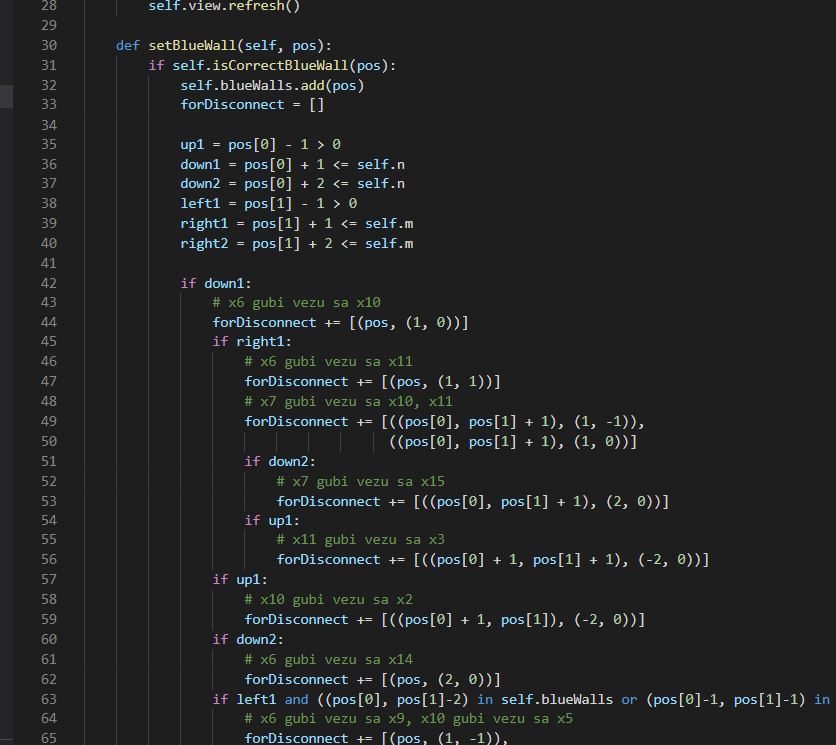


*Слика 10 Функција која ограничава кретање пешака*

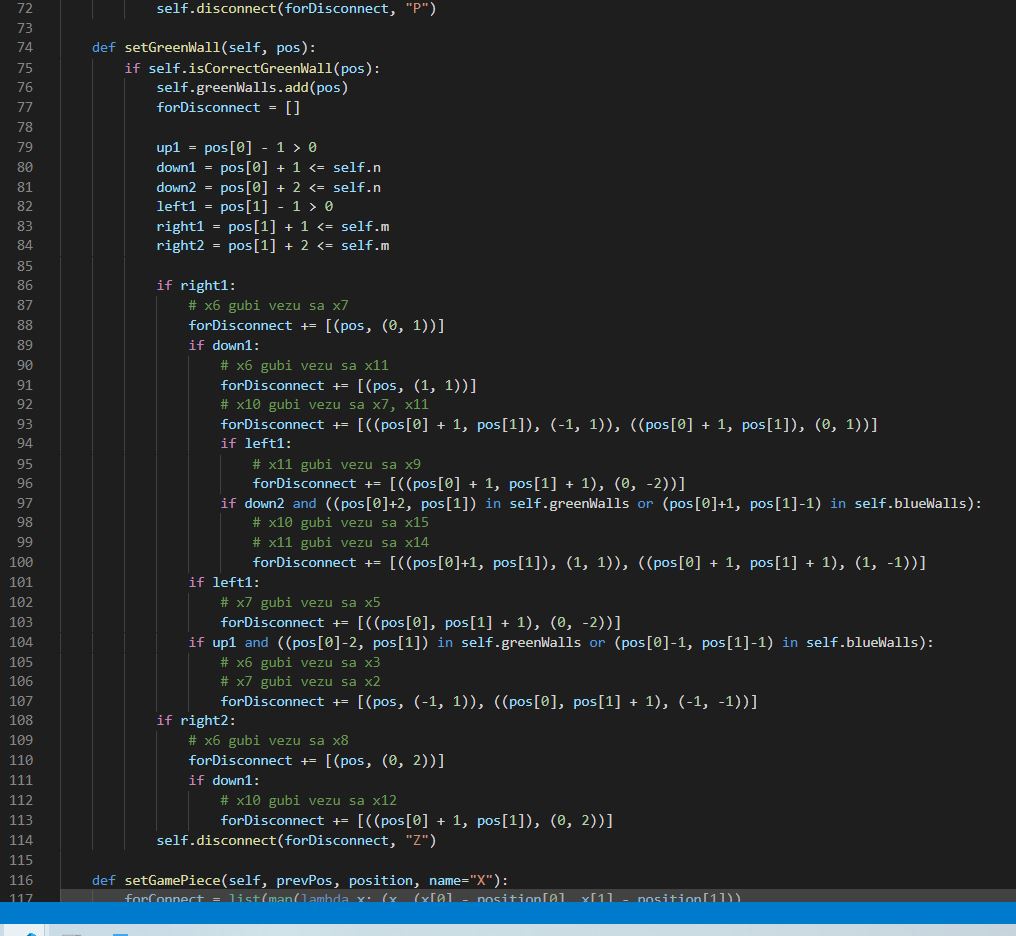
На слици 11 може се видети изглед функције *move* класе *Player.* Повратна вредност функције је стање у којем се фигура налазила пре одигравања новог потеза играча. Ако играч остане без зидова потез може да садржи само одабир фигуре и нову позицију фигуре, што се дефинише променљивом *wall*. У случају да играч још увек поседује зидове у зависности од прослеђене боје зида укупан број зидова те боје се декрементира за 1.



*Слика 11 Функција којом се мења стање играча*

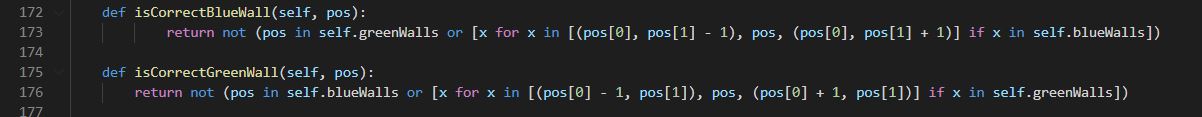


*Слика 12 Функција за постављање плавих зидова*



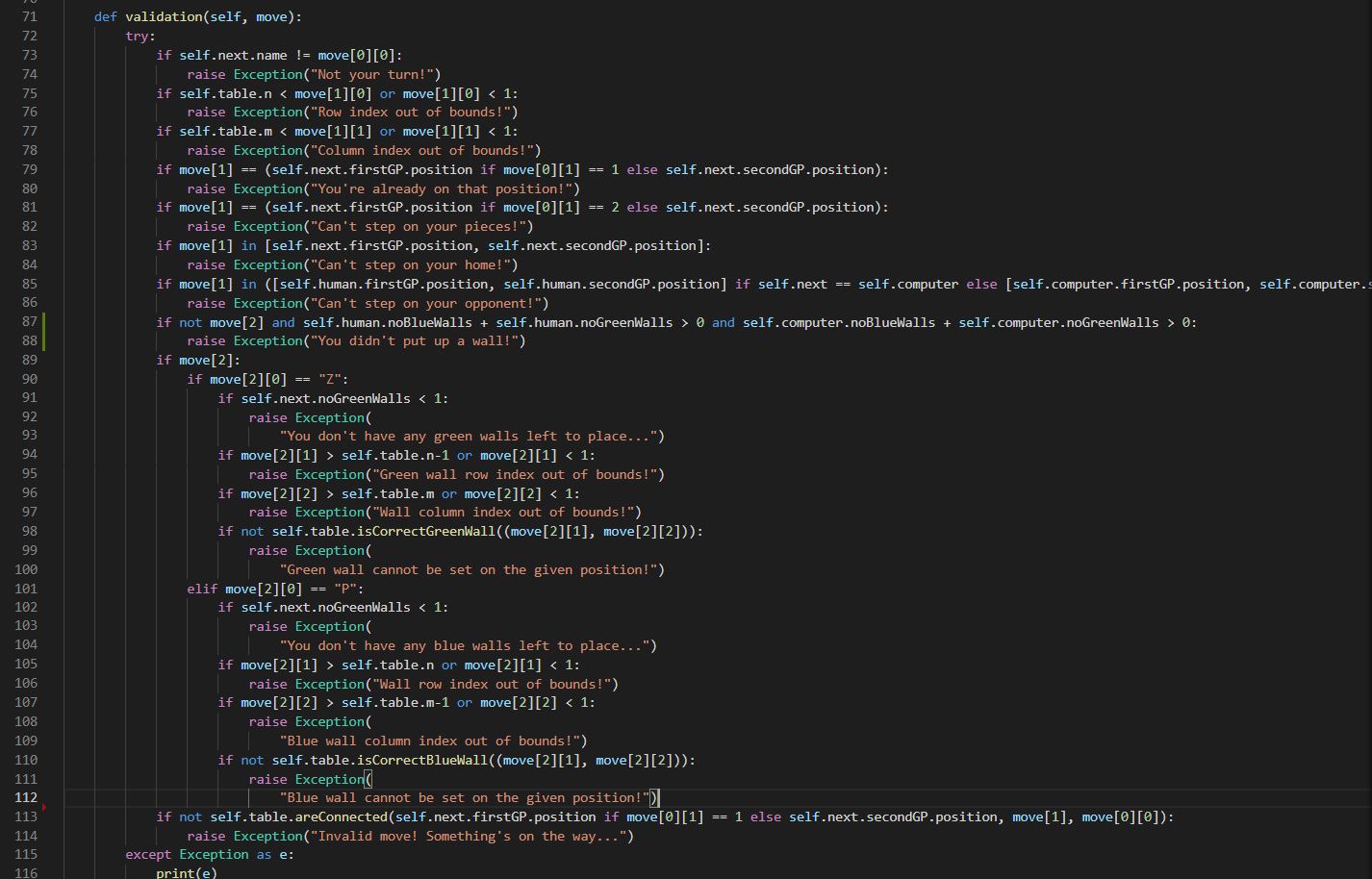
*Слика 13 Функција за постављање зелених зидова*

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 слици 13. Остатак код коришћеног у поменутим функцијама биће описан у некој од каснијих фаза израде пројекта, због тога што није релевантан за дефинисање проблема постављених фазом 1.

*Слика 14 Функције за испитивање валидности нових позиција зида*

Дакле функција *isCorrectBlueWall* враћа вредност True уколико се прослеђенa позицијa (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.

Такође, испитивање валидности сваког потеза (померај пешака, позиције зидова, унос потеза, …) одрађено је унутар функције *validation()* и то се може видети на слици 14.



*Слика 14 Функција валидације потеза*