**1. Постановка на проблема. Цели и задачи. Мотивация.**

Основната цел на този проект е да се изгради имплементация на играта с думи „Скрабъл“, която може да се играе от двама или повече човека или интелигентни компютърни играчи. Ще се направи опит за имплементация на тази игра, въз основа на която ще се правят допълнително персонализиране, като например да се промени размера на игрална дъска и подредбата на нейните плочки, да се използват множество речници, да позволява игра между различни на брой играчи и други. Дипломната работа ще обхваща основно английския набор от правила. Основният въпрос, на който трябва да се отговори е може ли играта да се имплементира отново с нови технологии, позовавайки се на съвременни проучвания, реализации и налични техники и могат ли да бъдат подобрени с помощта на наличната информация?

Забележка: Към момента на разписване на дипломната работа проектът вече е завършен или ще претърпи минимални промени. Единственият раздел, който е написан преди началото на разработката е „Теоретичен преглед на изходните с целта на проекта решения, технологии и средства за имплементация“. Дипломната работа е замислен така с цел да опише процеса на работа по правдоподобно и за да може по лесно читателя да проследи този процес.

**2. Теоретичен преглед на сходни с целта на проекта решения, технологии и средства за имплементация.**

**2.1. История**

„Скрабъл” е една от най-старите игри на думи, предлагани на пазара, създадена от Алфред Мошер Бътс от Poughkeepsie, Ню Йорк през 1933 г. [9]. Тогавашните игри с думи, като кръстословици и анаграми, не са имали система за оценяване, което, според него, ги възпрепятствало да бъдат толкова популярни, като например игрите с карти. Затова той измислил „Скрабъл“, който по това време се е наричал „LEXIKO“, а в по-късни години „CRISS CROSS“. Оценяването на буквите в играта е измислено от Бътс, като е изучавал първите страници на „Ню Йорк Таймс“. Това оценяване не се е променило оттогава. Играта на Бътс е била отхвърлена от големите производители на игри, докато идеята му не била приета от предприемача Джеймс Брунот и тогава играта получила окончателното си име „Скрабъл“, запазена марка през 1948 г. Първите няколко години били борба за Брунот, тъй като компанията била създала, до този момент, 2400 екземпляра на играта и претърпяла загуби от 450 долара. Но постепенно популярността на играта нараснала. Твърди се, че президентът на „MACY” е научил за играта и си я закупил, което довело до нейната популярност. Оттогава играта е променила собственика си, но не е загубила от популярността си. По-долу е даден пример за игра на „Скрабъл“ в процеса на игра.



*Фигура 1. Пример за игра на „Скрабъл“. „HAPPY” е изиграна първа дума, като други думи са прикачени към „HAPPY“, за да образуват допълнителни думи.*

Приблизително 150 милиона комплекта от играта са продадени по целия свят и има около 4000 клуба около света, които се занимават с нея [11]. Според статистиките, играта е популярна и подлежи на търсене. Що се отнася до софтуерни имплементации на играта, съществуват много такива, които носят със себе си нови функционалности и способи за интелигентна компютърна игра. Търговският софтуер за „Скрабъл” е много търсен. Според доклада от Марк Ричардс и Еял Амир [7], търговски софтуер „CrossWise“, е довел до печалби от 3 милиона долара в продажби.

**2.2. Правила**

Правилата са заимствано от официалната уеб-страница на „Scrabble Hasbro” [9]. Играта се играе на дъска с полета 15х15, като някои полета съдържат специални съкращения като DL, TL, DW, TW (двубуквени, трибуквени, двойни -дума, тройна дума), които от тук нататък ще наричаме „първокласни полета“.

* Играта започва със 100 плочки, поставени в торбичка. Торбичката съдържа 27 вида плочки, 26 от които са като всяка буква от английската азбука, а 27-мата е празна плочка, която може да действа като всяка буква.
* Всеки тип плочки може да бъде повторен няколко пъти, например буквата „Е“ трябва да бъде намерена върху 12 плочки, докато буквата „Q“ се намира само на 1 плочка. Всяка плочка има прикачен към нея номер, който представлява увеличаване на резултата за играча със съответната стойност на номера, ако играч използва плочката за да извърши хода си. Празните плочки са на стойност 0 точки.
* Всеки играч трябва да изтегли на случаен принцип равен брой плочки от торбичката, като този брой обикновено е 7. Всички плочки, които играчът има, се съхраняват в поставката на играча. Плочките, останали след като играч изиграе своя ход, се наричат „остатък на поставката“.
* Стартиращият играч трябва да направи валиден ход, като постави съответните плочки в стартовото положение на дъската, отбелязано със звездата в центъра на дъската. Валидността на думата се проверява с речник, преди тя да се играе (при игра на турнири, невалидни ходове може да се накажат чрез добавяне на точки към противниковия резултат).
* Думите могат да се поставят на дъската само по хоризонтален или вертикален начин. След като се изиграе дума, стойността на всяка плочка от тази дума се добавя към резултата от хода, който се добавя и към общия резултат на играча. Ако на първокласно поле се играе дума, резултатът за тази дума или отделна буква може да бъде увеличен.
* След първия ход на играта всяка следваща дума трябва да бъде прикрепена към плочка, която вече е на дъската. Прикрепяването може да бъде към всяка част от думата (началото, средата и край), стига да е част от валидна дума. За да е валиден хода също се изисква изиграните плочки да образуват валидни думи с всички съседни плочки, към които са свързани.
* За всяка играна плочка от поставката на играча, играчът трябва да изтегли еднакъв брой плочки от торбичката. След като торбичката е празна, играчите могат да продължат само с плочките, които имат на поставките си. Играчът може да използва своя ход, за да подмени плочки от поставката си, стига те да са равни на брой на плочките, от които той иска да се откаже.
* Използването на всичките седем плочки в от поставката на играч в един ход води до „бинго“, което добавя допълнителни 50 точки към резултата на играча.
* Играта приключва, когато играч е изиграл всичките си плочки, без да има останали в торбичката или когато и двамата играчи пасуват своя ход (поради невъзможност да играят дума).

**2.3 Предизвикателства**

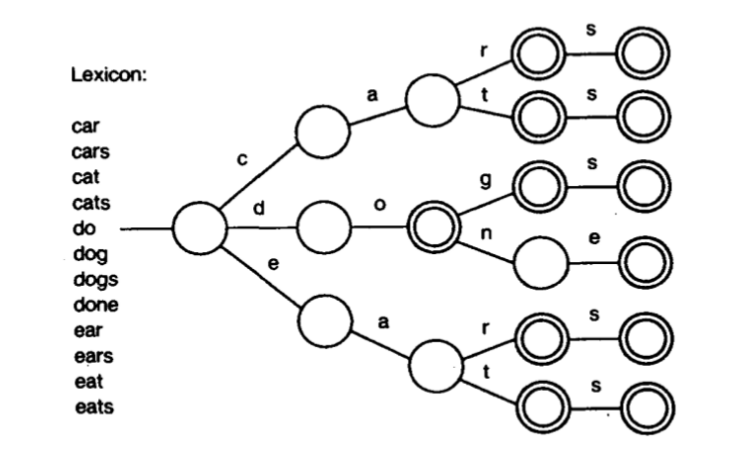
Едно от основните предизвикателства при разработването на компютърни игри е създаването на достоен „AI“ (изкуствен интелект) противник. В историята на играта е имало проучвания от различни учени и с тези публично достъпни знания е разработен „AI“ наречен „Maven“, който все още не е претърпял загуба в турнир от световно ниво [4]. Струва си да разгледаме какви техники „Maven“ използва, за да бъде такова предизвикателство за хората играчи да го победят и да видим кои от тях могат да бъдат внесени в текущия проект.

Друго предизвикателство е да се разгледат проучвания, които обхващат най-добрите алгоритми за валидиране на игра на думи и генериране на ходове. Използването на неправилни механизми може да доведат до състояние, при което „AI“ противника да не вземе в предвид множество възможни ходове, да играе ходове с ниски резултати или да му отнема твърде много време, за да вземе решение. Има безброй начини, по които една игра на „Скрабъл“ може да се разиграе, затова е важно AI поритвника да има достатъчно свобода да изследва различни възможности и да взема интелигентни решения, но и достатъчно ограничения, за да взема по бързо решения в играта. Maven използва вариант на съществуващите, добре проучени алгоритми, които са в основата на повечето съществуващи реализации и които ще разгледаме в следващите раздели.

**2.4. Алгоритъма на Апел и Якобсън**

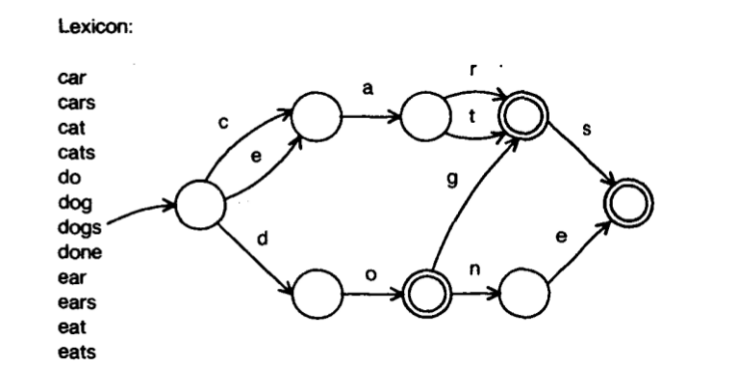
Важно научно откритие при разработката на AI противник е разгледано в статия публикувана от Апел и Якобсън [2]. Целта на учените е била да разработят ефикасен рекурсивен алгоритъм, който да се използва от програма за бързи изчисление, способна да играе играта „Скрабъл“. Една от основните технологии на програмата е била „DAWG“(насочен ацикличен граф от думи).

Едно от предизвикателствата на алгоритъма е съхраняването на лексикона на играта в паметта. За своя проект Апел и Якобсън използват лексикон с 94 240 думи. Един от начините, по които те успяват да го приложат, бил в „trie” от 117,150 възела („trie” е структура от данни наподобяваща дърво в компютърните науки и поради начина, по който се използва, може да се нарече и синтактично дърво). Всяка буква има начален възел и отделните клонове са свързани към тези букви за всяко различно продължение (фигурата по-долу).



*Фигура 2. Лексикон и съответното му „trie“. Заградените възли са крайни възли, т.е. те* *представляват края на валидна дума.*

Те открили, че това „trie“ е било доста обемисто, тъй като заемало около 780 Кбайт. Това, което направили, било да представят това „trie“ като „DAWG“, в който всички повтарящи се под-дървета са обединени в едно. Това намалило броя на възлите до 19 834, а използваната на памет до 175 Kbytes. По-долу е даден пример.



*Фигура 3. Пример за DAWG. Възли с ръбра, водещи до същото състояние по отношение на валидни словообразувания. Различни пътища, водещи до това състояние, са слети.*

Експериментът е извършен на компютър VAX 11-780, който с максимална памет 64MBs [8] е много по-малко в сравнение със съвременните машини. Тъй като повечето съвременни машини, вече имат гигабайти памет, представянето на речника като „trie” не би довело до проблеми с недостиг на паметта.

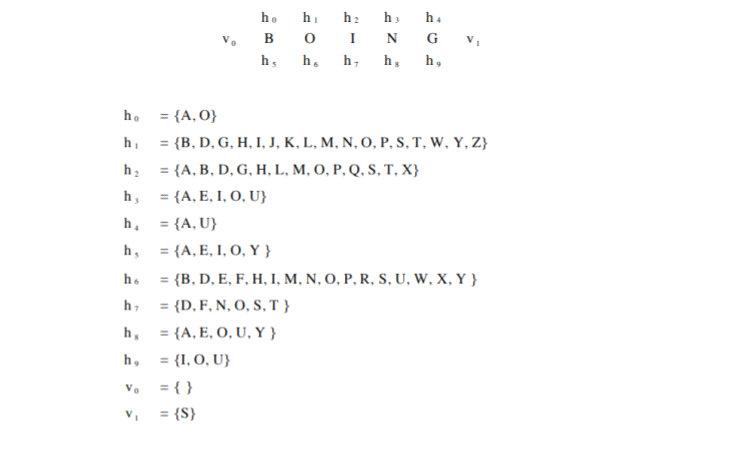
Друго предизвикателство било да се реши проблема - как да се намерят всички легални ходове, като се имат в предвид текущите плочки на дъската? Алгоритъмът, който учените разработили, се състоял от няколко техники.

**2.4.1. Техники**

Една от използваните техники е да се ограничи проблема до едно измерение. Играта на думи вертикално е като игра на думи хоризонтално, но на транспонирана дъска. Това позволява алгоритъмът да бъде проектиран за игри с хоризонтални ходове, които той лесно би се адаптирал за вертикални игри.

Друга техника са т.нар. „кръстосани проверки“. При извършването на хоризонтален ход, всяка поставена плочка трябва да образува валидна дума, когато е свързана с вече поставените плочки, които са вертикално съседни на нея. Но основна забележка е, че при хоризонтални ходове само една плочка може да се добави към отделните колони. Знаейки това не е твърде трудно задача да се изчислят всички валидни кръстосани проверки, което е важно за оптимизиране на цялостното генериране на ходове от AI противника и намаляване на броя на проверките за валидни думи.

Трета техника е използването на т.нар. „котви“. Котвите са полета от дъската, които са в съседство с вече поставени плочки на дъската. Според правилата на „Скрабъл“, всяка нова дума трябва да бъде свързана със съществуваща, което означава, че всяка дума на дъската (освен първата) е използвала котва, която я свърза със съществуваща дума. Използвайки тази информация, алгоритъмът започва да търси валидни ходове, като локализира всички котви на дъската, и специално се опитва да локализира тези, които са разположени в най-лявото съседно пространство на поставена плочка, което служи за отправна точка за генериране на ход. Следната фигура служи за изясняване на котвите и кръстосаните проверки.



*Фигура 4. Буквите „h” и „v” представляват котви за текущите плочки и възможни разположения за букви. Буквите, които биха преминали кръстосаните проверки, са дадени под думата. [2]*

След откриване на котвите, алгоритъмът започва генериране на легални ходове, изчислявайки валидни леви части на дума (представки), поставени в поле, което е котва и изчислявайки всички десни части на дума (наставки) за всяка валидна представка. Представките са изградени или от плочки от поставката на играча, или от плочки, които вече са на дъската, но не и двете едновременно. След генериране на всички ходове за дадена представка, тя се подменя за друг и в процеса на генериране на валидни ходове размерът на представката ще нарасне. Ако използваната структура е „DAWG“, валидни ходове се изчисляват чрез обхождането му използвайки плочки от поставката на играча или плочки, които вече са на дъската. Всяко успешно обхождане на структурата, което завършва с краен възел, се счита за валидна дума.

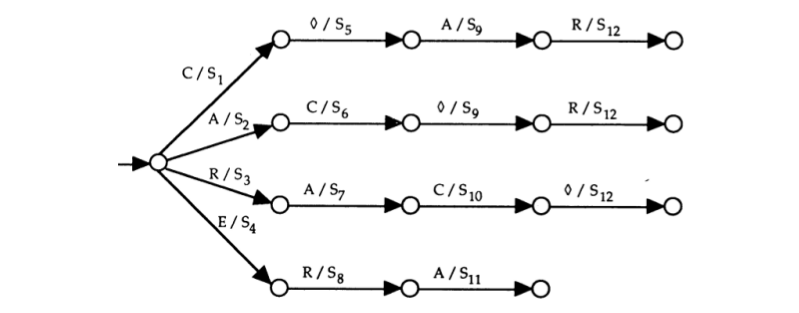
**2.4.2. Анализ**

Алгоритъмът на Апел и Якобсън е най-разпространеният в имплементациите на играта „Скрабъл“. Използван е от много други учени, благодарение на ефективните си техники. „Maven” също използва този алгоритъм [4]. Тъй като компютрите днес са много по-мощни от тези преди 30 години, различните структури от данни за съхранение на лексикона може да се окажат по-достъпни и по-лесни за използване, но техниките за транспониране, извършване на кръстосани проверки и използването на котви все още са много ефективни за решаване на задачата, поставена от играта „Скрабъл” и най-вероятно ще се използват във финалната имплементация на този проект.

Важно е да се отбележи, че този алгоритъм по никакъв начин не е стратегически. Единствената (но голяма) полза е бързината, с която взема решение за най-добрите възможни ходове. Той извършва само еднократно търсене и, в крайна сметка, избира хода оценен като най-добър в настоящата ситуация и не взима в предвид, например, че използването на избраната дума може да даде възможност на опонента да използва първокласно поле и не прави анализ на вероятността опонента да се възползва от тази възможност, тъй като не се правят симулации. „Maven” е една такава система, която използва този алгоритъм и също така използва и други стратегически елементи, които го карат да играе подобно на човешки играч.

**2.5. Алгоритъмът на Стивън Гордън**

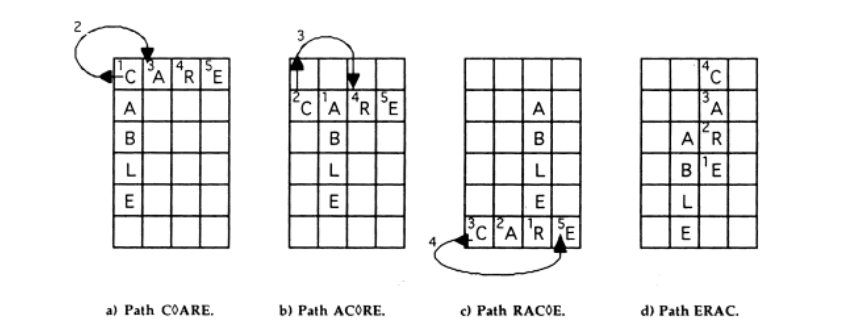
Гордън предлага реализация използваща двупосочен „DAWG“, наречен „GADDAG“, който работи на сравнително обратен принцип на „DAWG“. Алгоритъмът на Гордън представя всяка дума в даден лексикон в няколко варианта, които се добавят към графи (все още не е минимизиран). Броят на вариантите на дадена дума е равен на броя на буквите в тази дума. Всеки вариант на думата е разделен от специален знак, наречен „разделител“, който разделя думата на представка и наставка. Но има една особеност в реда на буквите, особено в представките. Пример е даден по-долу, последван от обяснение.



*Фигура 5. Пример за представянето на думата „CARE” чрез използването на „GADDAG” на Гордън. Ромбоидът служи за разделител, който разделя представката и наставката. Буквите в представката са в обратен ред поради посочената по-долу причина.*

**2.5.1. Техники**

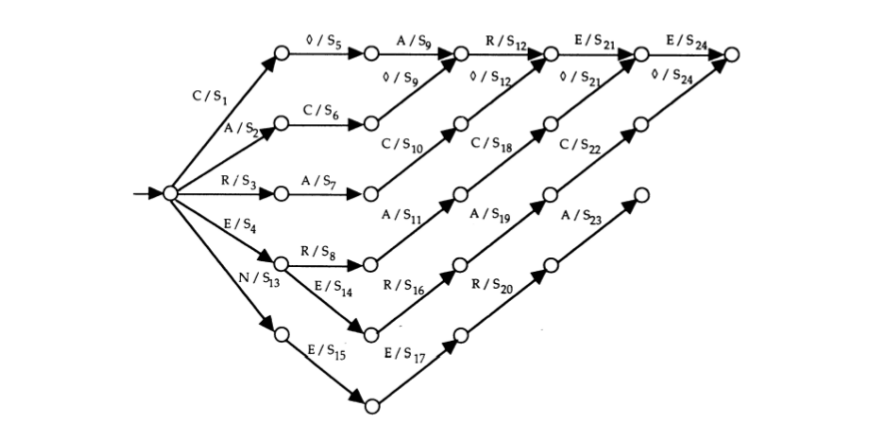
Алгоритъмът на Гордън използва същата техника като този на Апел и Якобсън по отношение на избора на котва. Подобно на техния алгоритъм, дума се изгражда чрез изграждането на представка първо вляво от котвата и наставка отдясно на котвата. В техния алгоритъм всички генерирани представки (използвайки плочките на поставката на играча) се поставят пред закачената буква и се проверяват дали образуват дума (размерът на представката се увеличава, докато не се достигне границата на дъската или не се достигне до вече поставени плочки). В алгоритъма на Гордън, подобно на алгоритъма на Апел и Якобсън, наставката е изградена в посока, обратна на котвата (вдясно), но и представките са изградени в посока, противоположна на котвата (вляво). Буквите в представката, които се намират преди разделителя в една „GADDAG” структурата са обърнати, защото те се прикрепят в посока, започваща от котвата и отдалечаваща се от нея вляво. Пример е даден по-долу.



*Фигура 6. На фигура „a)” „Генератора на думи” е настроен да премества добавянето на букви вляво от даденото поле. Буквата „С” е добавена първо на даденото поле. След като се достига разделителят и затова генератора превключва изграждането на думата вдясно от зададеното поле. На фигура „b)”, „ A” се поставя на даденото поле и „C” се добавя вляво от „A“, образувайки „CA“. След като разделителят е достигнат, „R” се добавя вдясно от полето, а „E” се добавя вдясно от „R“, образувайки „CARE“. Идеята е същата и за останалите фигури.*

Защо „GADDAG” е по-бърз при генериране на възможни ходове от „DAWG“? За пример нека приемем, че на дъска хоризонтално е поставена думата „HARPY” [10]. Ако играчът иска да използва буквата „P“ като отправна точка и има интервал от две букви над нея, използвайки „DAWG“ алгоритъмът ще трябва да премине през всички думи в лексикона, за да намери думите, които имат „P“ като трета буква. При „GADDAG“ машината ще види всички думи, които имат „P“ някъде в него и ще премине през всички клони на „P“ за да формира думи. Това е благодарение на разделителите, които позволяват всяка дума, съдържаща „P“, да бъде представена с начална буква „Р“ и комбинация от представки и наставки. Например, „GADDAG” може да съдържа думата “EXPLAIN” като „PXE <> LAIN” и да се стигне до заключението, че „EXPLAIN” е валидна дума. В „DAWG” машината ще стигне до това заключение, едва като посети клона, започващ с „E” и установи, че третата буква на думата е „P“. При „GADDAG” това „лутане” е силно ограничено.

„GADDAG” структурата в горния пример може да се минимизира. Важно е да се отбележи, че „GADDAG” е начин за представяне на една дума представлява в няколко варианта, като я разделя на различни представки и наставки. Някои варианти ще имат наставка с по-голяма дължина от наставката на други варианти, но като се има предвид, че крайният резултат е един и същ, това означава, че по-дългите наставки са съставени от по-късите наставки на други варианти, заедно с още няколко букви. Например в „CAREEN“, „C <> AREEN” и „AC <> REEN” имат наставки „AREEN” и „REEN“. Когато разглеждаме тези наставки можем да кажем, че „REEN” е част от двете наставки. Това означава, че „C <A“ и „AC <>“ водят до същото състояние, от което „REEN“ е валидна наставка, като по този начин вместо да има две отделни състояния за „C <> A“ и „AC <>“, структурата може да бъде сведена до минимум чрез сливане на състоянията, към които водят „C <> A“ и „AC <>“, тъй като еднакви. Структурата може допълнително да се съкрати, като се вземе „RAC <> EEN“, който има наставката „EEN“. Използвайки същата логика „RAC <>” може да се комбинира със състоянието, до което водят „C <AR” и „AC <R“, тъй като и трите състояния биха довели до използването на наставката „EEN“. Фигура, изобразяваща съкратената структура, е показана по-долу.



*Фигура 7. Под-графика на минимизиран „GADDAG” за думата „CAREEN“. Състояния, които имат еднакви наставки са обединени.*

**2.5.2. Анализ**

Въпреки че разликата в използваната памет може да е незначителна в съвременните машини и различията в скоростта могат да бъдат драстично намалени, най-голямото предимство на алгоритъма на Гордън ще стане явно при симулирането на възможни ходове. Симулациите, както ще се види в следващите раздели, включват извършване на ход, генериране на евентуален ход, който може да изиграе противника и след това да се генерира най-добрия ход в отговор на противника. Това би представлявало едно двуетапно и четириетапно генериране, което би снабдило AI играча с ходове, които са по-добри за спечелване на играта, но такива многоетапни генерирания са за сметка на памет и скорост. Играчите в турнирни игри обикновено имат ограничение във времето от 25 минути. Затова генериране на ходове с двойно по-голяма скорост от една „DAWG” структура е една безценна функционалност. Също така в разглежданата литература никъде не е споменато какви са изискванията спрямо хардуер за турнири. Единственото изискване към AI играча е да работи коректно. Алгоритъмът на Гордън е добра алтернатива на този на Апел и Якобсън. Това дали той е по-добър или по-лош, зависи от изискванията на играта и наличните хардуерни ресурси. Ако имаме изискване откъм скорост, предпочитаният вариант е „GADDAG“, тъй като той е по-бърз и ако изискване е по-ефективно използване на ресурсите на машината, то „DAWG“ е предпочитания вариант, тъй като той използва по-малко памет.

**2.6. Maven**

Доклад за напълно работещия „Maven” е изготвен от неговия създател Брайън Шеппард през 2001 г. [3]. Първите му опити да произведе AI играч са през 1983 г. и след период на развитие от няколко години системата му получава сегашното си наименование и претърпява допълнителни подобрения. „Maven” разполага с богат набор от функции и параметри за оценяване на ходове и може да извършва симулации, за да оцени средните резултати от различни сценарии. Като цяло, „Maven” извежда резултат за състоянието на играта, като взима в предвид поставката си с плочки, плочките, които вече са на дъската и текущия резултат на играта. Въз основа на тази информация, „Maven” определя най-добрия възможен ход, който до голяма степен зависи от следните евристични функции или подходи.

**2.6.1. Евристики**

Първият евристичен подход е т.нар. „баланс на гласни/съгласни“. Важна част от това да се играе оптимално е да се избягва това да оставаме с поставка от плочки, с които е трудно или невъзможно да се играе. Поставката се счита за такава, ако има много повече гласни, отколкото съгласни или обратното. Наличието на твърде много от двете прави по-трудна задачата да се образува дума и да се направи ход. Наличието на баланс прави тази задача по-лесна и би довело до по „плавно” извършване на ходове. Подхода има прагова стойност, която текущата поставка трябва да премине. Поставката ще има положителна стойност, ако има добър баланс на букви и отрицателна, ако едните букви превъзхождат броя на другите. В този случай „Maven” ще даде приоритет на ходове, които биха използвали от преобладаващите букви, за да се улесни в бъдеще [11].

Втория подход е т.нар. „U-With-Q-Unseen“. Оказва се, че буквата „Q” е доста трудна за използване във валиден ход и ако това да я имаме на поставката ни, може да направи играта с тази поставка по трудна. Ако обаче на поставката има буквата „U“, „Maven“ ще се постарае да запази тази буква и да я играе с „Q“, тъй като това е комбинация, която прави извършването на ход с „Q“ по-лесна задача.

Третия подход е т.нар. „First-Turn Openness“. Общоприето е, че играчът не трябва да позволява на другия играч да играе дума, използвайки първокласни полета. Използвайки дълга дума като първа в играта, шансът на противника да направи ход, с който да се възползва от първокласно поле е по-голям. Поради тази причина, когато „Maven“ е първи на ход в игра на „Скрабъл“, той винаги ще започне играта с къса дума, като например двубуквена такава. Това намалява възможността противникът да спечели предимство, използвайки първокласно поле.

„Maven” използва тези подходи по различен начин и използва различни стратегии за генериране на ходове в зависимост от етапа, в който се намира текущата игра. В даден момент, играта се счита да е в една от три възможни фази.

**2.6.2. Фази на играта**

Първата фаза на трае между началото на играта и до момента, в който останат 16 непознати плочки (9 в торбичката и 7 на поставката на противника). В тази фаза на играта генерирането на ходове е едноетапно и метода за оценка на възможните ходове е достатъчно бърз и точен, че да се играят доста добри ходове.

Втората фаза е преди края на играта и продължава от момента, в който има 16 непознати плочки, до момента, в който торбичката е празна. „Maven” използва т.нар. функция „oracle“, за да извърши двуетапно търсене за най-добър ход. Шеппард твърди, че при извършването на ход обикновено се използват около 4.5 плочки. Това означава, че извършването на търсене с повече от два етапа е неоптимално, тъй като поставката на играча държи до 7 плочки и промяната на поставката се случва бързо. „Maven“ прави симулация, като изиграва няколко хода със случайни поставки и избира да направи ход с висок средна разлика в точките. Идеята за подобни симулации идва от дискусиите на Шепърд с националния шампион Рон Тикерт през декември 1987 г. В собствените си експерименти, Рон е имал избора да играе или „AWA“, или „AWARD“ като начален ход. „AWARD“ очевидно би му дало повече точки, но след 50 игра, започвайки с една от двете думи, Рон стигнал до заключението, че „AWA“ носи по-високи резултати в дългосрочен план. Това помогнало на Шеппард да развие евристиката за „First- Turn Openness“ и да осъзнае на значимостта на симулациите.

Третата фаза е в края на играта. По време на тази фаза играта се превръща в игра с „перфектна информация“, т.е. възможно е да се определи какви са плочките на противниковата поставка, тъй като торбичката е празна и всички останали плочки са на дъската или на поставката на „Maven“. „Maven“ използва т.нар. алгоритъм за търсене „B\*“, за да избере най-добрия ход. Шеппард, също така, тества и „Алфа-бета“ алгоритъма, но той се оказва твърде неефективен в края на играта, тъй като изчисляването на възможните сценарий може да се окаже доста сложно (до 14 етапа) и симулациите с такава дълбочина биха били непрактични.

**2.6.3. Анализ**

„Maven“ е система, която е итеративно разработена за дълъг период от време, използвайки техники, които са добре проучени. Видно е също, че Шеппард и неговият екип са добре запознати с игрите на професионално ниво и са поддържали връзка с професионални играчи с цел да подобрят системата си. Най-добрата игра, в която е участвал „Maven“, според Шеппард, е срещу Адам Логан в тяхната 12-та игра, разиграла се в „AAAI-98“ турнира, където „Maven“ обърнал дефицит от 98 точки в победа с последния си ход, като изиграл думата „MOUTHPART", на стойност 100 точки.

Като цяло „Maven” е едно амбициозно решение за игра, използвано на професионално ниво, но и ни дава доста полезна информация, разписана в следните точки:

* Алгоритъмът на Апел и Якобсън е приложим и е доказан като добър, тъй като се използва на професионално ниво.
* „GADDAG” на Стивън Гордън също е добра алтернатива, тъй като Шеппард споменава, че вероятно е по-добър от „DAWG“, но не е толкова универсален.
* „Скрабъл” е игра позоваваща се на непълна информация до края на играта. Предвиждането на възможни сценарии използвайки алгоритми като „A \* и „алфа-бета” разпадане е неоптимално, тъй като различните сценарии произхождащи от състоянието на дъската и плочките на поставките на играчите са твърде много. Алгоритъма „B \*” обаче е a добро решение.
* Използваният алгоритъм може да се подобри чрез имплементиране на основните евристики на „Maven“, като баланс на гласни/съгласни или „First-Turn Openness“.
* Симулациите, с повече от два етапа, рядко ще доведат до по-добри резултати и необходимото изчислително време е непрактично.

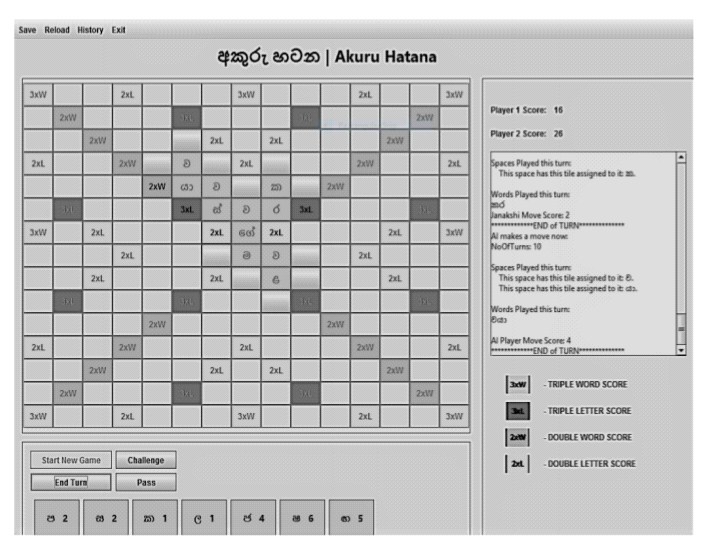
Също така има голяма неяснота по отношение „Maven“, тъй като няма предоставен „open-source“ код и наличния документ не споменава нищо конкретно относно имплементацията му. За щастие обаче, има няколко реализации подобни на „Maven“, разработени от други учени, и техните изследвания, макар и непълни, разкриват някои неясноти относно развитието на умен AI играч, подобен на „Maven“.

**2.7. Допълнително проучване и имплементации**

Има различни имплементации и доклади от други учени и всички те предоставят нови идеи при изследване на проблема или реализация на решение.

**2.7.1. Изпълнението на Д. Джанакши Дуланга**

Една реализация е от Д. Джанакши Дуланга [1], която проектира игра на „Скрабъл” базирана на сингалския език. Ключови моменти в нейната реализация са добавянето на език, който не е официално разпознат, използвайки структурата „GADDAG“ за търсене на възможни ходове, използването на „алфа-бета“ разбиване за симулации в крайната фаза на игра, съхраняването на дъската в масив. Играта е разработена с помощта на „Java” с „MVC“ модел. Крайния резултат може да се види по-долу.



*Фигура 8. Реализация на играта „Скрабъл“ на „Java“, снабдена с „Java GUI“, сингалските букви и дневник на извършените движения вдясно.*

**2.7.2. Откритията на Стивън Гордън относно потенциалната евристика**

В документ от Стивън Гордън [5] се говори за разликите между вероятностното търсене и претеглената евристика в играта. Документът разкрива информация, която не е налична в документа на Шеппард, отнасяща се към изчисляването на евристики на базата на останалите на поставката букви. Също така предоставя формула за изчисляване на съотношението гласна/съгласна като „VCMix = MIN (3V + 1, 3C) L = MIN (3V +1 - L, 2L – 3V)“, където „V“ е броя на гласните, „C“ е броя на съгласните и „L“ е броя на буквите. Също така, в документа се говори за евристика, свързана с дублирани плочки на поставката и как се оценяват. Също така, се споменава за една доста проста формула за избор на най-добър ход след симулации и тя е следната - „резултат на текущия ход - резултатът на противника за следващия ход + резултатът на следващия ход на текущия играч“.

**2.7.3. Откритията на Марк Ричардс и Еял Амир относно прилагането на интелигентни симулации**

В своите открития, Марк Ричардс и Еял Амир [7] говорят за симулации и това как, тъй като броят на плочките в торбичката намалява, плочките в поставката на противника стават по-предвидими. Тази информация може да се използва при извършването на симулация, за правене на предположение какви плочки може да има противникът на поставката си, като това предположение би било по-точно, тъй като торбичката е по-празна. Това води до по-предвидими потенциални ходове от противника, което помага на AI играча в избора на подходящ ход, тъй като ще знае какво да очаква. В края на играта AI ще има пълната информация за това какви ходове може да очаква, тъй като ще знае всички плочки на поставката на противника.

**2.7.4. Откритията на Фрей Коноли и Даяна Грен относно прилагането на различни приоритети и стратегии за генериране на движения**

Фрей Коноли и Даяна Грен [6] предлагат една доста интересна имплементация на играта. Тяхното решение използва „trie” за представяне на шведски лексикон, което не се представя като „DAWG” или „GADDAG“. Това показва, че реализация е възможна без използването на тези структури от данни. Съшо така, те са разработили три различни AI играча с различни начина на игра. „High Score Word” играчът е еквивалентен на този от Апел и Якобсън. Той извършва ходове, където думата с най-висок резултат се играе без допълнителни евристики. „Bonus Square” играчът дава приоритет на ходовете, които използват някоя от първокласните полета, тъй като те, въпреки че може да не донесат най-висок резултат, ще попречат на противника да ги използва. „Balance on Rack” играчът дава приоритет на ходовете, които оставят след себе си поставка, балансирана откъм гласни и съгласни, което позволява на AI играча по-лесно да играе ходове в бъдеще. Учените накрая открили, че „Bonus Square” играча печели най-много игри.

**2.8. Заключения от теоретичния преглед**

Има и други добре документирани реализации на играта [2, 11, 12, 13], които повече или по-малко представят подобни идеи. От проучването оставаме с една доста добра представа как би протекло имплементирането на подобно решение. Има достатъчно доказателства за ефективността на основните алгоритми, с което стигаме до заключението, че те могат да бъдат използвани като отправна точка за разработката на интелигентен AI противник. Откритията ще ни послужат при изграждането на един базов начин за генериране на ходове, който по нататък ще се обогати чрез използването на евристики и симулации. Тъй като в разгледаните реализации се използват технологиите „Java” и „C” („Maven“), изборът на „C #” като език за програмиране не би трябвало да се окаже проблематичен за игра като тази. При реализацията на този проект, първо ще се разработи функционалността на играта, след което всички визуални елементи.

**3. Проект на предлаганото софтуерно решение**

**3.1. Избор на език за програмиране и фреймуърк („framework“)**

От разгледаната литература става ясно, че има много приложими езици и технологии, които могат да бъдат използвани. Това означава, че няма реални ограничения откъм избор на инструменти за разработването на подобно приложение. Например, решението на Д. Джанакши Дуланга е написано на „Java“, който е програмен език на високо ниво.

Тази дипломна работа не обхваща техническите характеристики на разликите между езиците за програмиране. Вместо това езикът е избран въз основа на наличните технологии за разработка на приложения, които той предлага и поради лични подбуди. За имплементацията на този проект бяха избрани „C #” и „.NET Framework“.

**3.2. „C#” и „.NET Framework“**

„C #” е език за програмиране на високо ниво и е разработен през 2002 г. от екип на „Microsoft“, ръководен от Андерс Хейлсберг [12]. Той е разширение на езика „С“, който е език на ниско ниво, но между двата езика има мнго различия. „C #” се базира на „.NET Framework” на „Microsoft“. „.NET Framework” предоставя инструменти и библиотеки за разработване на уеб приложения или приложения за „Windows” машини и телефони [13].

Комбинацията от „C #” и „.NET Framework” много често се използва за разработването на уеб приложения, базирани на т.нар. „форма” („form“). Едно приложение се счита за такова, когато има обект (формата, която има колекция от под-обекти) и той променя състоянието си в зависимост от въведената от потребителя информация и някаква логика (обикновено наречена „бизнес логика“, т.е. набор от стъпки, зададени от доставчика на формите), която извършва обработка на въведената информация.

Този процес на работа на формите може да бъде адаптиран за играта „Скрабъл” като гледаме на играта като на една форма и при работата с тази форма се следва набор от стъпки:

* На играчът (потребителя) му се представят дъска, торбичка с букви и поставка за буквите, като те са част от формата.
* Играчът играе дума от поставка си (вход) и обявява, че извършва ход (изпраща входа).
* Играчите или трета страна (сървър, т.е. доставчик на формуляр) потвърждават въвеждането (проверява дали хода е валиден).
* Ако хода е валиден, той се съхранява и играта дава ред на друг играч (формата се актуализира и състоянието на играта се променя).
* Предишните стъпки се повтарят.

Това са основните стъпки на играта. Има и други стъпки между тях (като преначертаване на дъската, пропускане на ход или получаване на грешка от доставчика на формата), но всички те работят по един и същи начин - информация се въвежда от потребителя и в зависимост от този вход формата променя или запазва състоянието.

**3.3. Уеб приложение („Web application“)**

Предимство на уеб приложенията е, че те се инсталират само и единствено на една машина и това е сървъра (доставчик на формуляри). Играчите (потребителите), които биха искали да играят, се нуждаят само от уеб браузър и адреса за достъп до приложението. Това прави приложението достъпно за по-широка аудитория поради минималните изисквания за употреба. Също така, нито една от реализациите от прегледа на литературата не е уеб-базирана. Вместо това те са инсталирани на машината на разработчика, което ги прави достъпни само за него.

Друго предимство е, че уеб приложенията могат да бъдат достъпни както от десктоп потребители, така и от потребители на мобилни устройства. Въпреки че е възможно да се разработи имплементация за една единствена платформа (като например за „Windows” / „iOS” / „Android“), това би направило приложението зависимо от тази платформа. Уеб приложението е достъпно за всички платформи, стига да имат уеб браузър. Това ни позволява да се фокусираме повече върху самата логика на играта и не толкова върху подробностите, свързани с дадената платформа.

**3.4. „ASP.NET” и „ASP.NET Core“**

„ASP.NET” е „open-source” фреймуърк, използван за разработката на сървърни уеб приложения [14]. Последната официална актуализация на „ASP.NET“ е от октомври 2017 г., тъй като от тогава „Microsoft“ полага повече усилия за разработването на наследника на „ASP.NET“, а именно „ASP.NET Core“. Той е пренаписана и подобрена версия на „ASP.NET” и е проектиран да бъде независим от използваната платформа и да предлага по-добра производителност от своя предшественик.

Има и други разлики между „ASP.NET“ и „ASP.NET Core“, но фокуса на „Microsoft“ е в развитието на „ASP.NET Core“, тъй като това е по-нова технология, се е снабден с функции, които липсват при „ASP.NET“ или да не са толкова надеждни. Имайки предвид, че имплементацията на играта „Скрабъл“ ще следва стандартния работен процес за разработката на едно уеб приложение базирано на форми и че целта на тази разработка е да се установи какво е възможно с инструментите и информацията на разположение, „ASP.NET Core“ е избран като фреймуърк на този проект.

Приложенията, базирани на „ASP.NET Core“, обикновено следват определен модел при разработка и този модел се нарича „MVC” („Model-View-Controller“).

**3.5. „MVC“**

MVC е архитектурен модел, който обикновено се използва за разработване на приложения, които имат потребителски интерфейс [15]. Целта на модела е да раздели приложението на три основни компонента с цел поддържането на разделение на задачите между тях, преизползване на код в тях и паралелната им разработка. Трите компонента са т.нар. “модел, изглед и контролер”.

**3.5.1. Модел („Model“)**

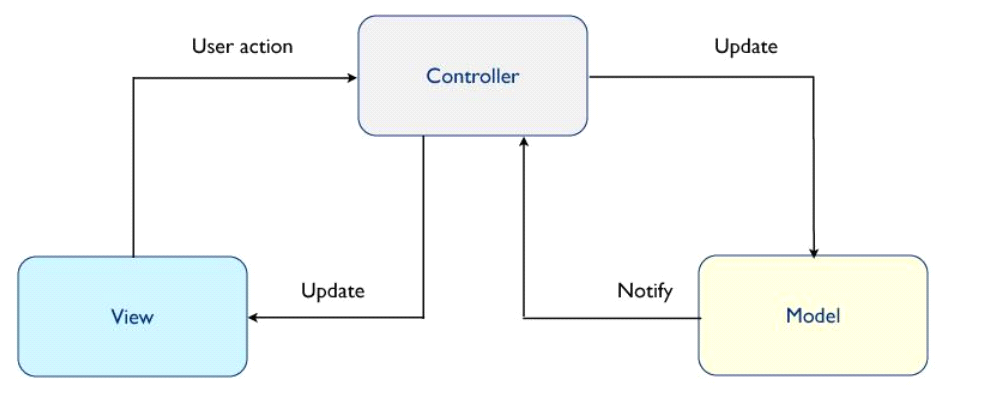
Моделът представлява обект или структура от данни в приложението. Също така, моделът съдържа логика и правила за използването му и за управление на данните, които съхранява. В тази имплементация моделите са всички обекти, които се съхраняват в кода на играта.

**3.5.2. Изглед („View“)**

Изгледът е графичното представяне на информацията. Показаната информация може да бъде персонализирана под формата на диаграма или графика или от каквото се нуждае бизнеса и информацията може да произхожда от модел. В тази имплементация, изгледа се използва като графичен интерфейс на играта, тъй като той показва дъската, играчите, резултатите, поставките с букви, плочките и други. Изгледът е разработен като една „Razor“ страница, в която се комбинира HTML и „C #“ код, чрез които се визуализира играта.

**3.5.3. Контролер („Controller“)**

Контролерът приема вход от потребителя и го използва за да промени модел или за да генерира изглед. В тази имплементация има един контролер, който управлява всички взаимодействия между потребителя и играта.



*Фигура 9. Демонстрация на взаимодействието между модела, изгледа и контролера в едно приложение.*

**3.6. „Full Stack” разработка**

Повечето уеб приложения се състоят от три основни компонента. Компонентите са клиент, сървър и база данни [16]. Изхождайки от примера за едно приложение базирано на форми, клиентът е потребителят, който предоставя вход на сървъра, който действа като доставчик на форми. След като доставчикът валидира въведените от потребителя данни, те се съхраняват в „склад” за информация, наречен база данни. Когато потребителят направи заявка към доставчика на форми да му се предостави дадена форма, доставчикът търси заявената форма в базата и я връща на потребителя. След като потребителят актуализира формата, тя се изпраща обратно на доставчика и той съхранява актуализираната форма в базата за използване в бъдеще. Също така, потребителят може да създава нови форми или да изтрива съществуващи (стига да има право).

**3.6.1. Клиент**

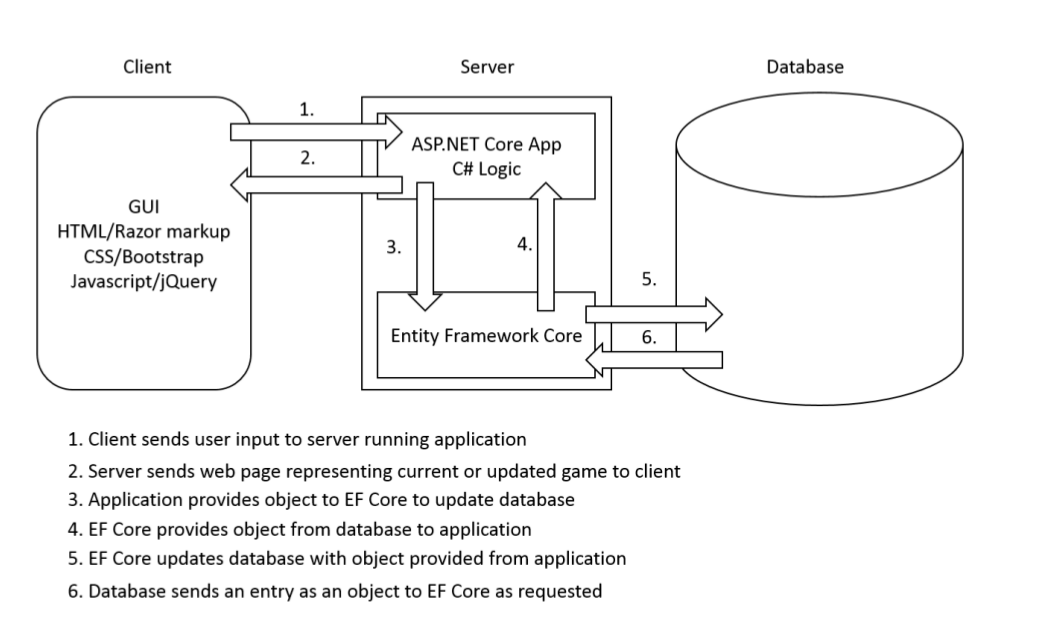
Клиентът представлява платформа, която използва комбинация от технологии, които са необходими на потребителя за да взаимодейства с приложение. Тези технологии създават „front-end” частта на приложението и също така се наричат „клиентски технологии“. В тази имплементация тези технологии са „HTML” (генерирани от „Razor Pages“), „CSS” (написан лично и предоставен от „Bootstrap“) и „JavaScript” (написан лично и предоставен от „jQuery“). „HTML” и „CSS” са отговарят за стилизирането на уеб страницата, а „JavaScript” е отговорен за логиката, която се случва на страницата. Въжмоностите на „Bootstrap” допринасят за стилизирането на страницата, а „jQuery” позволява по-лесна манипулация на елементите на страницата.

**3.6.2. Сървър**

Сървърът е платформа, която използва комбинация от технологии, които са отговорни за изпълнението на логиката върху формата с информация, предоставена от клиента (логиката е невидима за клиента). Тези технологии създават „backend” частта на приложение и също така се наричат „сървърни технологии“. Сървърът е отговорен и за обработката на клиентските заявки и само той има право да прави промени в базата данни на приложението. В тази имплементация контролера събира информацията, въведена от клиент, и извиква различни обекти (в случая модели) или помощни методи в кода на приложението, за да изпълнява логиката.

**3.6.3. База данни**

Базата данни е платформа, която използва комбинация от технологии, които са отговорни за съхраняването на специфични за потребителя и специфични за приложението данни за бъдеща употреба. Обикновено базата данни се съхранява на отделен сървър и съдържа таблици и редове, които представляват обекти или модели на приложението. „ASP.NET Core“ предоставя „IIS Express“, който при стартиране, генерира „Microsoft SQL Server“, който действа като хранилище за базата данни и позволява на сървъра да взаимодейства с нея. Технологията, която позволява по-лесно взаимодействие между сървъра и базата данни, се нарича „EF Core” („Entity Framework Core“). Тя служи за генериране на обекти в кода, като ги обвързва с данни от базата или за съхранението на обекти в базата [17]. Генерираният елемент може да се използва за попълване на страница, която потребителят вижда. Тези операции се случват чрез използването на „Context” обект, който имплементира функционалността на „EF Core“.



*Фигура 10. Пример за работния процес в едно уеб приложение, използващо клиент, сървър и база данни.*

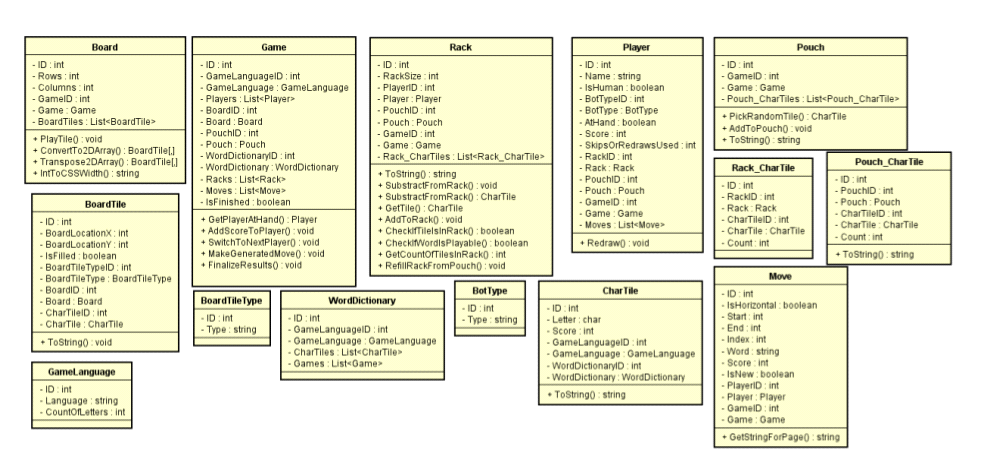
**3.7. Проектиране**

Всички разгледани до този момент подточки послужиха за запознаване на читателя с технологиите, използвани в този проект и как те работят заедно, за да образуват цялостно приложение. Тази подточка ще обясни как тези технологии и инструменти са адаптирани към разработеното „Скрабъл” приложение. Ще се представят първоначалните идеи по отношение на структурата и дизайна на приложението и как тези идеи се доразвиват в окончателната версия на проекта. Необходимо е да се отбележи, че няма да се навлиза в подробности относно логиката на отделните компоненти, което ще е тема на следващия раздел.

**3.7.1. „Backend“, модели и дизайн на базата данни**

Като се има в предвид структурата и работния процес на едно „fullstack“ уеб приложение и дадения пример за една форма, се стигна до заключението самата „Скрабъл“ форма да се представи като модел в „MVC“ приложението и всяко пропърти („property“), на тази форма, също да се представи като модел. Това е с цел да може всяко пропърти да претърпява логическа обработка в приложението, посредством неговия модел. Освен това всяка инстанция на даден модел ще се съхранява в базата данни. Това ще позволи на потребителите да изпълняват промени върху играта и тези промени да са достъпни и за останалите играчи. Това създава един цикъл на получаване на текущото състояние на играта, актуализирането й с нови данни, които от своя страна ще се подадат на друг потребител, който също ще актуализира данните й и т.н. Предимството на базата данни е, че потребителите не е необходимо да пазят при себе си текущото състояние на играта. Всяко ново зареждане на играта в браузъра би им предоставило най-новата версия на играта.

Бяха разписани множество чернови с идеи за структурата на моделите преди и по време на разработването на играта, като в крайна сметка те придобиха следния си вид.



*Фигура 11. Моделите, представени като таблици в базата данни.*

Важно е да се отбележи, че напълно правилна структура за моделите не съществува. Предложеният дизайн, макар и не идеален, е достатъчно добър, за изпълнението на този проект.

По-долу е кратко описание на всеки модел.



*Фигура 12. Индивидуални описания на моделите*

Моделите бяха разработени първи в приложението, като след това бе генерирана база данни, която да съхранява техните инстанции.

**3.7.2. Server-database communication**

Комуникацията между сървъра и базата данни се осъществява чрез обекта „ScrabbleContext“, който имплементира функционалността на „EF Core“. „ScrabbleContext” позволява на сървъра да изтегля записи от базата и да ги представя като обекти, и обратното, да пази обекти като записи в базата. Не е необходимо да се използват стандартните „SQL” заявки.

**3.7.3. Комуникация клиент-сървър**

В съвременните уеб приложения клиентите изпращат „HTTP” заявки към сървъра, за да поискат и извършат промени върху форми. Сървърът изпраща „HTTP” отговор, указващ дали заявката е била успешна или не [18]. Това приложение работи на същия принцип. „HTTP” заявката и отговорите съдържат информация относно дадено действие. Например при заявка за текстов файл от страна на клиента, в отговора си сървърът може да каже, че текстовият файл е наличен или че той не е достъпен. В това приложение заявките съдържат информация относно играта и промени в нея. Ако потребителят постави плочки на дъската и натисне бутона „Submit“, изпратената информация ще съдържа буквите, които потребителят е играл и къде ги е играл. Сървърът ще използва тази информация и ще изпрати отговор на потребителя, относно това дали ходът е грешен (поради невалидна английска дума например) или ще изпрати на потребителя информация за новото състояние на играта, която клиентът ще използва, за да актуализира страницата за потребителя. Заявките и отговорите се изпращат и получават чрез „AJAX” („Asynchronous JavaScript and XML“) заявки от клиента, които се създават от библиотеката „jQuery“, която приложението използва от страна на клиента.

**3.7.4. „Frontend” дизайн**

Приложението е разработено като т.нар. „single-page” приложение. Това означава, че когато клиента извърши промяна в играта и сървърът върне променената игра, страницата, на която клиентът е в момента, се актуализира с новата версия на играта, вместо да се зарежда нова, предварително създадена страница. Този дизайн се използва, за да се намали броят от необходимите за разработка уеб страници до една единствена и да се предостави едно по-приятно изживяване за потребителя. Този тип актуализация е възможна чрез използването на „AJAX“.

В това приложение „AJAX” се използва за изпращане на информация от клиента до сървъра и за изчакване на отговор, без да се напуска текущата страница. Докато клиента чака отговор, всички елементи на страницата се заключват и след получаване на отговор, съдържанието на страницата се актуализира. Това всичко се случва без да се напуска текущата страница, което води до по-добро потребителско изживяване.

**4. Детайлно описание на предлаганата софтуерна реализация.**

**4.1. Пример за игра на „Скрабъл” в текущия проект**

Ето как изглежда една игра във финалната версия на това приложение.



*Фигура 13. Финалната версия на играта*

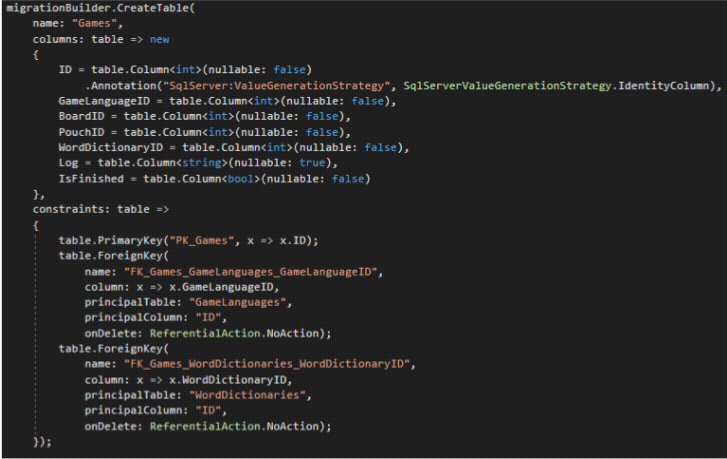
Тъй като приложението използва „MVC“ модела, логиката от страна на клиента, от страна на сървъра и базата данни е разработена паралелно. Базата данни е източникът на данни, затова първото нещо, което трябва да се направи, е да се попълни базата с тестови данни (от предварително създадените модели и техни инстанции), с които да се работи. Тъй като такава база няма, първо трябва да бъде създадена.

**4.2. Създаване на базата данни**

„EF Core” предоставя необходимите инструменти за автоматичното създаване на базата данни, стига моделите и техните пропъртита да са налични, като създава таблици базирани на тези модели и пропъртита. При създаване базата и таблиците, „EF Core” също взима в предвид, кога чужд модел се използва като пропърти в даден модел и отразява тази зависимост в таблицата. Например, ако обектът „Game“ има списък с обекти тип „Player“, всеки запис „Players“ таблицата ще има пропърти, наречено „GameID“, дори и да няма свойство „GameID“ в „Player“ модела. Имайки пропъртитата „GameID“ и „Game“ в „Player“ модела, можем да получим идентификационния номер и инстанцията на „Game“ обекта, към който се отнася „Player“ обекта. Това извличане на чужди инстанции и как то работи се разглежда по подробно в подточката „Lazy Loading“.

За да се попълни базата данни с модели, които се представят като таблици, командата „EntityFrameworkCore / Add-Migration Initial“ се въвежда в „Package Manager“ конзолата. Тази команда създава т.нар. „миграционен“ файл, който съдържа всички команди за генериране на база с таблици и конфигурира някои от полетата в таблиците да съдържат „foreign key“, ако се отнасят към други модела. Командите са с „C #” синтаксис, а не „SQL” синтаксис. Следващата използвана команда е „EntityFrameworkCore / Update-Database“, която създава базата и таблиците в нея въз основа на миграционния файл. Миграционния файл може да се създаде с допълнителни команди за вмъкване на записи в базата (т.нар. “засяване” на базата), като записите се намират в предварително създаден „Context” файл. Засяването и „Context” файла са разгледани в следващата подточка.

Следната фигура е фрагмент с код от миграционния файл, генериран от „EF Core“. Фрагментът създава таблица с полета, като някои полета биват „primary key” и „foreign key“.



*Фигура 14. Фрагмент с код от миграционния файл, генериран от „EF Core“.*

**4.3. Засяване на базата данни със „ScrabbleContext” файла**

Файлът ScrabbleContext съдържа клас, който имплементира интерфейса наречен „DbContext” от „EF Core” и може да предостави инстанция на обект. Инстанция на „ScrabbleContext” може да се използва навсякъде в рамките на приложението, където сървърът иска да си комуникира с базата данни, което се случва главно в контролера или някои логически части на приложението. Командата „Update-Database” използва инстанция на „ScrabbleContext“, за да генерира базата и да я попълни с данни, чрез изпълняване на командите от миграционния файл. Данните, с които ще се попълни базата, са предоставени в метода „OnModelCreating()” на класа „ScrabbleContext“. Обектите в „OnModelCreating()” се добавят към миграционния файл при изпълнение на командата „Add-Migration” и в крайна сметка се добавят към базата с командата „Update-Database“.

Следната фигура е фрагмент с код от метода „OnModelCreating()” в класа „ScrabbleContext“. При изпълнение на фрагмента, инстанции на „Pouch” модела ще се добавят в таблицата „Pouches“.



*Фигура 15. Фрагмент с код от метода „OnModelCreating()” в класа ScrabbleContext*

Много от записите, които трябва да запишат в базата, разчитат на съществуването на други записи, преди те самите да могат да се запишат. Това рекурсивно изискване спира при записването на „CharTile” обектите. Таблицата на „CharTile” никога не се редактира и е необходимо тя предварително да се попълни с информация от разработчика. Повечето записи в другите таблици могат да бъдат генерирани автоматично при създаване на нова игра и при предоставяне на достатъчно информация за типа игра, която искаме, но таблицата „CharTiles“ винаги ще е такава, каквато е. Това е базата, върху която се изграждат всички останали таблици и записи.

Забележка: Функционалността за създаване на игри чрез потребителски интерфейс не е имплементирана. За да се създаде нова игра, е необходимо „ScrabbleContext“ файла предварително да се попълни с данни или да се добавят записи в базата чрез използването на „SQL Server Object Explorer“ на „Visual Studio“.

**4.4. Комуникация клиент-сървър-база данни**

Когато клиентът достъпи уеб страницата на приложението (чрез адрес в браузъра) или извърши някакво действие върху нея, се изпраща „HTTP” заявка до контролера на приложение. Контролерът съдържа множество методи и в зависимост от получената заявка ще се извика съответният метод. Извиканият метод, в повечето случаи, ще зареди един „Game” обект от базата данни, ще направи промени по него, ще подаде промените към базата и ще изпрати отговор на клиента, който е изпратил заявката. Отговорът може да е съобщение за грешка или актуализираните данни на „Game” обекта, с които да се обнови страницата.

Забележка: „Game” обектът, който се зарежда, се съхранява в методите на контролера (той е т.нар. „hard-coded).

**4.5. Извличане на записи от базата данни като обекти**

Предоставяйки типа и идентификационния номер(„ID“), или всякаква друга информация отнасяща се към даден запис в базата към „ScrabbleContext“ обекта, „EF Core“ намира записа с дадения идентификатор или пропъртита в дадената таблица и връща обект, чийто тип е този на таблицата, от която е изтеглена информацията (напр. изтеглянето на запис от „Games“ таблицата ще върне „Game“ обект). Пропъртитата на върнатия обект ще бъдат със същите стойности, с които са били в базата.

Има една подробност обаче. Например, в „Game“ модела имаме „BoardID“ и „Board“ модела като пропъртита. „BoardID“ е просто целочислено („integer“) пропърти , но „Board“ е пропърти, което има свой собствен запис в „Boards“ таблицата, който от своя страна има модели като пропъртита и т.н. Тъй като „Board“ пропъртито не е а прост тип като цяло число, низ („string“) или „Boolean“, то в рамките на върнатия „Game“ обект ще бъде с нулева стойност („null“). Но имайки „BoardID“ пропъртито, ние знаем кой запис от „Boards“ таблицата ни е необходим и можем да изтеглим този запис едновременно с „Game“ обекта, вместо да го теглим отделно. Това става чрез техника нарича се „lazy-loading“.

Следната фигура е пример за извличане на „Game” обект от базата. Изтегленият обект съвпада с записа в таблицата, който има идентификатор 1.

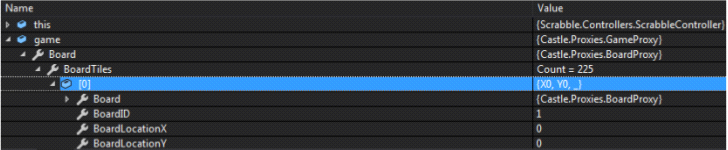


*Фигура 16. „*Game” о*бект, който се изтегля от базата данни.*

**4.6. „Lazy-loading“**

„Lazy-loading” позволява на „ScrabbleContext” обекта да изтегли обект от базата, както и обектите, които съдържа в една заявка [19] (техниката ни се предоставя чрез инсталиране на необходимия „NuGet” пакет). Например, когато изтегляме „Game” обект от съответната таблицата, обектът ще съдържа и пълен „Board” обект. Тази операция продължава в дълбочина, така че „Board“ обекта също ще съдържа пълен списък с „BoardTiles“ обекти, тъй като отделните „BoardTile“ обекти имат пропърти „BoardID“, което ги отнася към съответния „Board“ обект. Следователно, тъй като „Game“ знае (или евентуално ще разбере) за всеки друг обект, използван в приложението, чрез изтегляне на „Game“ обекта ние получаваме и всеки друг обект - дъска, плочки, играчи, движения и т.н. .Това ни спестява усилията да издърпаме всеки обект поотделно и после да попълним „Game” обекта с тези обекти.

Следната фигура разглежда „Game“ обекта от фигура 16 и ясно се вижда, че „Board“ обекта, както и неговата колекция от „BoardTiles“ обекти са били заредени и са част от „Game“ обекта.

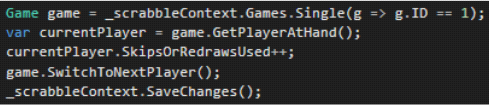


*Фигура 17. Разглеждане на* „Game” *обекта.*

**4.7. Запазване на промени в базата данни**

Тъй като повечето от обектите, които са претърпели промени поради логика извършена от сървъра, първоначално са изтеглени от базата данни, обектите вече са свързани с правилния запис в правилната таблица. По този начин отразяването на промените, направени на тези обекти, в базата се постига лесно с извикване на метода „SaveChanges()” на „ScrabbleContext” обекта. Този метод записва всички промени (включително създаване или триене) на обекти в базата. Ако не се извика този метод, тогава промените няма да се отразят в базата.

Следната фигура демонстрира как извлечен обект от базата се променя и как „SaveChanges()” се използва за запазване на всички промени на обектите в базата.

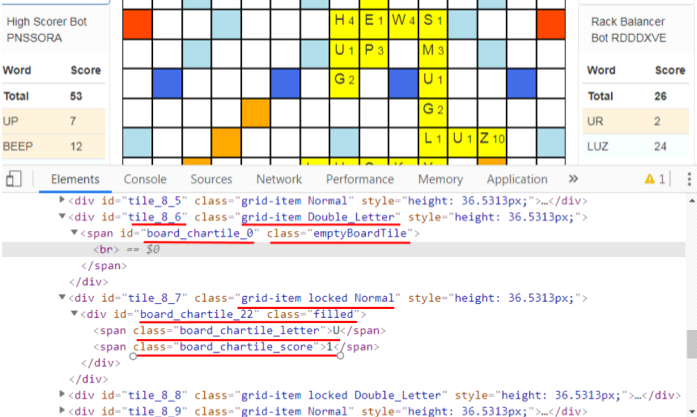


*Фигура 18. Пример за изтегляне на обект, промяната му и актуализация на базата.*

**4.8. Структура на уеб страницата**

Приложението се състои от една единствена страница, а съдържанието й, е отражение на „Game” обекта (заредено от базата) и обектите, с които е той е свързан (играчи, поставки, торбичка, движения и т.н.). Ако се погледне „HTML” кода на страницата, може да се види, че много от елементите имат свой идентификационен номер („ID“) и един или няколко класа. Тази информация се предоставя от „Razor“, който генерира „HTML“ страница с атрибути, които произхождат от „Game“ обекта. Това да имаме точни описания на всеки елемент на страницата е важно, тъй като когато потребителят изпраща въведените данни на сървър, заявката, която се изпраща, съдържа колекция от елементите на страницата, които са променени. С помощта на атрибутите на тези елементи, сървърът може да намери обектите със съвпадащи атрибути в „Game” обекта. Тъй като уеб страницата е отражение „Game” обекта, при промяната на този обект и съхраняването му в базата, следващия път, когато уеб страницата се актуализира, тя ще показва съдържанието на новата версия на обекта.

Следната фигура е фрагмент от „HTML” кода на уеб страницата. Подчертаните редове маркират структурата на плочката с буквата „U” и плочката отляво от нея. Атрибутите се използват за стилизиране, изпълнение на логика и подаване на данни от клиента към сървъра.



*Фигура 19. Фрагмент от „HTML” кода на уеб страницата.*

**4.9. Валидация**

При разработването на уеб приложения важна тема, която трябва да бъде обсъдена, е валидирането на въведените от потребителя данни. Има случаи, когато потребителят ще предостави неправилни данни, като например когато играе дума, която не е валидна или играе дума, която не е свързана с друга дума на дъската. Тези грешки трябва да бъдат прихванати в процеса на валидация, който може да се извърши или от страна на клиента, или от страна на сървъра, като валидацията, от която и да е от двете страни, има своите предимства и недостатъци.

Обикновено, когато потребителят използва браузър за достъп до уеб страница, браузърът зарежда уеб страницата, както и всички външни файлове, които са свързани със стила или логиката на тази страница. В това приложение се зареждат допълнителни „CSS” файлове и „Javascript” файл. „CSS” файловете отговаря за стилизирането на страницата, а „Javascript” файлът отговаря за логиката. „Javascript” файлът съдържа набор от функции, които се извикват всеки път, когато потребителят извърши действие, като например натискане на бутон. Тези функции са много на брой, но основната, която извършва валидирането, е тази, която се извиква при натискане на бутона за изпращане „Submit“. С натискането на този бутон потребителят се опитва да въведе нови данни в базата данни и затова сървърът (доставчик на формуляри) трябва да бъде сигурен, че данните са правилни, а не грешни или злонамерени.

**4.9.1. Валидация от страна на клиента**

Валидацията от страна на клиента се извършва на устройството на потребителя, посредством „Javascript” файлът, който се предоставя от сървъра при зареждането на уеб страницата. За да може клиентът да изпрати данни на сървър, данните трябва да преминат проверките, които се извършват чрез функциите в „Javascript” файла. Ако данните успешно преминат проверките се изпращат до сървъра и той ще запази промените без повторна проверка на данните (освен ако не се извърши и проверка от страна на сървъра). Ако данните са неверни, не се прави опит за комуникация със сървъра и вместо това потребителя получава съобщение за грешка.

Предимството при валидацията от страна на клиента е, че то е много по-бързо, тъй като проверките се съхраняват в уеб браузъра на клиента. В случай на грешка, потребителят не би трябвало да чака отговор от сървъра, за да разбере, че действията му са неправилни. Недостатъкът е, че логиката за валидиране се съхранява на клиента и потребителят, стига да знае как, може да модифицира валидацията или да промени данните, след като те са преминали през проверките и преди да бъдат изпратени до сървъра. С други думи, ефектът от валидацията може да бъде обезсилен и неправилни данни могат да бъдат изпратени до сървъра. Ако сървърът не извършва валидация от своя страна, промените по „Game“ обекта, които ще се запазят в базата данни може евентуално да „счупят“ играта. Това, също така, може да предпоставка да се разкрият пропуски в сигурността на сървъра.

Следната фигура показва валидация от страна на клиента. Опитът да се играят буквите „R“ и „H“ води до грешка, тъй като те не са свързани със съществуваща дума на дъската и също така не са свързани помежду си. Валидацията се извършва от страна на клиента и тъй като хода е грешен, сървърът няма нужда да се информира за действията на потребителя.



*Фигура 20. Пример за валидация от страна на клиента.*

**4.9.2. Валидация от страна на сървъра**

Валидирането на данни от страна на сървъра се извършва на машината, която съхранява самото приложение и го предоставя на клиентите. Когато клиент изпрати данни до сървъра, той ще извърши проверки върху тях, за да потвърди, че са коректни (хода е валиден). При валиден ход, „Game” обекта ще се актуализира и промените ще се съхранят в базата данни. В противен случай сървъра ще върне грешка на клиента.

Предимството на валидацията от страна на сървъра е, че потребителят няма достъп до сървъра и проверките, които се извършват на него. Ако потребителят изпрати грешни данни, той няма как да заобиколи проверките на сървъра и така вероятността да се счупи играта намалява. Друго предимство е, че няма необходимост от това, клиента да съхранява големи количества с данни, свързани с логика и валидиране. Недостатъкът е, че при всяко действие от страна на потребителя, ще трябва заявка да се изпрати до сървъра и де се изчака отговор. Това приложение си комуникира със сървъра чрез „AJAX“ заявки и тъй като страницата се заключва по време на една такава заявка, потребителят ще бъде принуден да изчака получаването на отговор, преди да може отново да взаимодейства със страницата. Това може да доведе до едно не толкова приятно изживяване за потребителя.

Следната фигура е пример за валидация от страна на сървъра. Опитът да се изиграе думата „HHEWS” води до грешка, тъй като тя не е валидна дума. Валидирането трябва да се извърши на сървъра, тъй като само той има речник с правилните думи, който се проверява за да установи коректността на думата. Речника не се съхранява при клиента, тъй като това би доведе до използването на повече от системните ресурси на клиента и също така, може да е предпоставка за злоупотреби от страна на клиента.



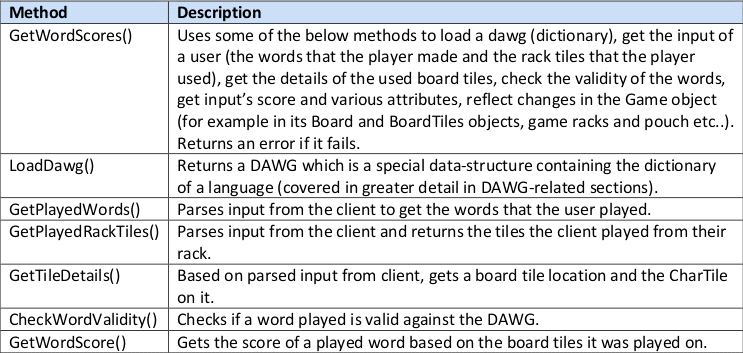
*Фигура 21. Пример за валидация от страна на сървъра.*

**4.9.3. Решение на проблема с валидацията**

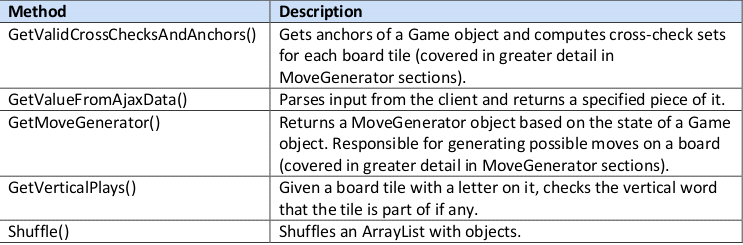
Това приложение извършва проверки както от страна на клиента, така и от страна на сървъра. Някои от проверките се повтарят и от двете страни, а други се извършват само на клиента или само на сървъра. Повтарящите се проверки включват валидиране, дали изиграната дума е свързана с друга дума на дъската, дали изиграната дума е първата дума в играта и други. Валидацията от страна на сървъра извършва тези проверки отново (в случай, че злонамерен потребител е подправил данните след преминаването на проверките от страна на клиента) и също така извършва и свои проверки върху думата, като например дали думата съществува (речника се съхранява на сървъра).

**4.10. „Helper” класът**

„Helper” файлът съдържа статичен клас, който държи в себе си много от методите използвани в приложението за валидиране на ходове, промяна на поставката, генериране на ход и т.н. Следните методи представя кратки описания на методите свързани със валидирането на ходове и оценяването им:

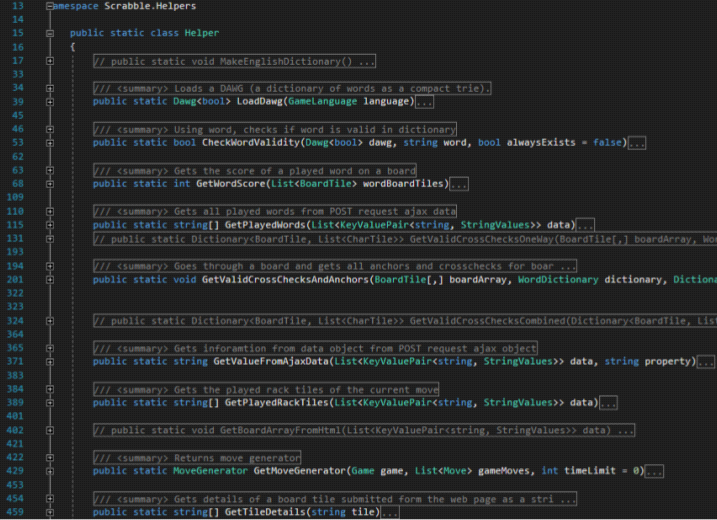


Следните методи са по-общи или се използват при генерирането на ходове:



„Helper” класът и методите му, както и методите, които се намират в моделите, обхващат по-голямата част от функционалността, необходима за да се играе играта от човешки играчи. Това включва потребител да играе дума, клиентът и сървърът да я валидират, сървърът да актуализира „Game“ обекта и базата данни, да предаде новото състояние на „Game“ обекта на клиентите, които от своя страна ще актуализират уеб страниците си.

Следната фигура показва фрагмент с код от „Helper” класа.

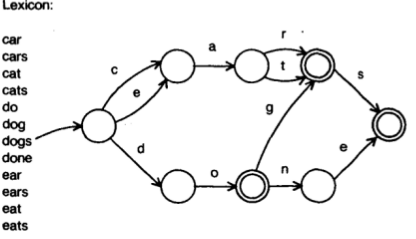


*Фигура 22. Фрагмент с код от „Helper” класа.*

**4.11. „DAWG“(„Directed Acyclic Word Graph“)**

Както стана ясно в прегледа на литературата, „DAWG“ е компресирана версия на едно „trie“ (дървовидна структура), която съхранява речника с думи на играта и в която повтарящите се звена (състояния) са обединени. Условието за обединение е две звена да имат едно и също продължение. Едно звено може да бъде крайно или не и да съдържа набор от думи. Едно крайно звено съдържа само валидни думи, в противен случай то не е крайно.

Следната фигура е пример за една „DAWG” структура. Звената с кръг във вътрешността са крайни. „CAR“, „CAT“, „EAT“, „EAR“ и „DOG“ се намират в едно и също крайно звено, тъй като и петте думи са валидни и могат да бъдат продължени с буква „S“. „С“ и „Е” са поставени в един и също звено, тъй като имат същото продължение с буквата „А“.



*Фигура 23. Пример за „DAWG” структура.*

**4.11.1. Пример за обхождане на „DAWG” структура**

В една „DAWG” структура валидацията на една дума става със скорост „O(n)” („n” е дължината на търсената дума). Това може да се обясни чрез следните стъпки, където питаме „DAWG” структурата дали думата „CAT” е в речника.

* Започвайки от корена (празна дума) на „DAWG” структурата се проверява дали има път от корена, обозначен с „C“. Тъй като все още няма дума, всички символи в азбуката са потенциални пътища. Пътят, означен с „C“, се следва, за да се достигне до не крайното звено, съдържащо думата „C“.
* Започвайки от звеното „С“, се проверява дали има път от това звено, който е обозначен с буквата „А“. Разглеждат се възможните пътища от звеното „С“, като в крайна сметка се следва пътя, обозначен с „А“, който води до не крайното звено с думата „CA“.
* Започвайки от звеното „CA“, се проверява дали има път от това звено, който е обозначен с буквата „T“. Разглеждат се възможните пътища от звеното „CA“, като в крайна сметка се следва пътя, обозначен с „T“, който води до крайното звено с думата „CAT“.
* Ако са били разгледани всички букви от дадената дума и е достигнато до звено, което държи в себе си тази дума, то думата ще се маркира като валидна, ако звеното, към което принадлежи, е крайно. В противен случай думата не е валидна.
* Ако не са били разгледани всички букви от дадената дума, то думата се счита за невалидна. Това се дължи поради липсата на възможни пътища от структурата, които да са обозначени с някоя от буквите на дадената дума, при обхождането на структурата. В най-бързия случай, една дума може да бъде маркирана като невалидна, когато се опитаме да последвате път от звено, което се предхожда от корена (който има пътища към всички букви).

**4.11.2. Защо бе избрана „DAWG” структурата пред „GADDAG” структурата**

„GADDAG” структурата, разработена от Стивън Гордън и описана от него като двупосочен „DAWG“. В прегледа на литературата, при сравнението на „DAWG” с „GADDAG“, станаха ясни предимствата и недостатъците на двете структури. Но тъй като и двете са добри структури, „DAWG“ структурата беше избрана за проекта, тъй като нейна имплементация беше налична като „NuGet“ пакет, наречен „DawgSharp“ и използвайки този пакет, времето за разработка на проекта драстично беше намалено.

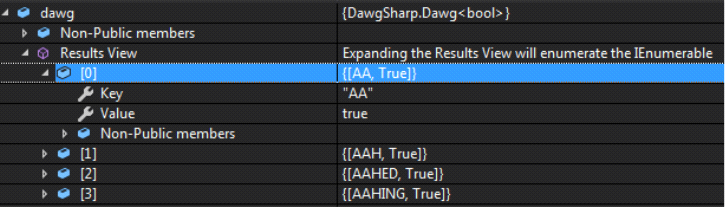
**4.11.3. „DawgSharp“**

„DawgSharp” пакета може да изгради „DAWG” структура използвайки текстов файл, съдържащ валидни думи [20]. Генерираната от пакета структура, е много подобна на „HashSet” (колекция от уникални елементи). Подобни са, тъй като и двете структури съдържат уникални думи и валидността на думата може да се провери, като се опитаме да я извлечем от структурата, използвайки думата като ключ (например извикването „dawg[“CAT”]“ ще върне „True“, а „dawg[“CATT”]“ ще върне „False“). Според страницата на автора на „DawgSharp“, тази реализация на „DAWG“, при съхранени 2 милиона думи в нея, използва само 2MB RAM.

Забележка: Тъй като сорс кода за „DawgSharp“ пакета не е наличен, не може да се докаже доколко тази „DAWG“ реализацията следва идеите на Апел и Якобсън, тъй като те използват компресирано „trie“, а „DawgSharp“ авторът използва „HashSet“. Въпреки това, реализацията на автора предлага полезни методи, чиято функционалност се базира на структура, която представлява компресирано „trie“, като това от Апел и Якобсън. Един много полезен метод е „MatchPrefix()“, който взема дума и връща всички думи от речника, които съдържат дадената дума като наставка. Този метод се използва в процеса на генериране на ходове за компютърния играч.

Речниците за играта са предоставени в „Helper” папката на проекта. Английският речник се намира във файла „englishWords.txt“.

Следната фигура е пример за „DAWG” структура, създадена от пакета „DawgSharp“. Структурата съдържа всички валидни думи и валидността на дадена дума може да бъде извлечена чрез използването на думата като ключ (ако съществува).



*Фигура 26. Пример за „DAWG” структура, създадена от „DawgSharp” пакета. В този пример думите „AA“, „AAH” и т.н. са валидни думи от речника*

**4.12. „MoveGenerator“**

„MoveGenerator” е обект, отговорен за съставянето на списък на всички възможни ходове, които може да направи играчът, като се има в предвид поставката му и текущото състоянието на играта и дъската.

В своя документ „Най-бързите структури от данни в света“, Апел и Якобсън говорят за една качествена структура от данни за съхранение на речник на играта „Скрабъл“, алгоритъм за генериране на всички възможни ходове за дадена игра и набор от предварително зададени ограничения за търсените от този алгоритъм думи и времето за намирането им. Тъй като използваната в този проект структура за съхранение на речника идва от от документите на Апел и Якобсън, е логично да се доразвие проекта с алгоритъма и техниките от същите тези документи.

За да се инициализира един „MoveGenerator” обект, първо трябва да му се зададат необходимите ограничения. Без ограничения алгоритъмът работи, като генерира всички възможни думи с всички възможни букви в азбуката от всички празни полета на дъската, като добавя към всяко свободно поле комбинации от букви вляво и вдясно. Също така, без ограничения на алгоритъма ще му е необходимо много време за да генерира всички възможни думи, при положение, че по-голямата част от тях, биха били думи, които нарушават правилата на играта. Производителността на алгоритъма може да бъде подобрена със следните ограничения:

* Вместо да се използват всички букви в азбуката, да се използват само буквите, които текущият играч има на поставката.
* Вместо да се използват всички празни полета на дъската, да се използват само тези, които са свързани с дума, която вече е на дъската (котви).
* Вместо да се използват всички букви от поставката на играча, да се използват само тези, които биха довели до валиден вертикален ход (да се спазват кръстосаните проверки).
* Докато алгоритъмът генерира хоризонтални ходове, едновременно с това да генерира и ходове върху транспонирана версия на дъската, за да се получат и всички вертикални ходове за оригиналната (не транспонирана) дъска.

**4.12.1. Техники за ограничаване на алгоритъма**

**4.12.1.1. Транспониране на дъската**

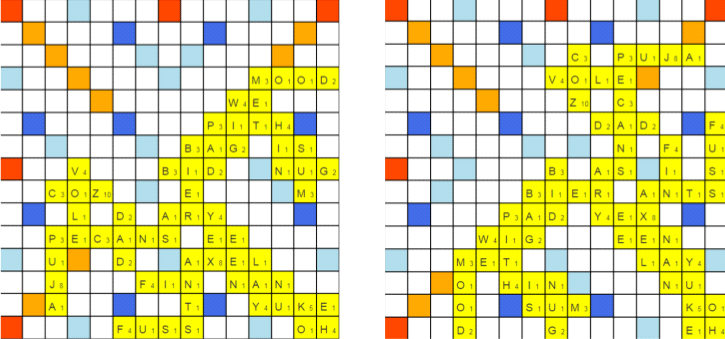
Тъй като алгоритъма на Апел и Якобсън генерира само хоризонтални ходове на дъската, за да накараме алгоритъма да генерира и вертикални ходове, приложими към оригиналната дъска, трябва да му дадем той да работи с вертикална версия на тази дъска. За целта се използва транспониране.

С налична двуизмерната матрица, транспонирането е начин за предоставяне на нова версия на матрицата, където редовете се показват като колони и колоните се показват като редове [21]. Тъй като дъската може да се счита за матрица, транспонирането й ще ни даде дъска, където всички хоризонтални думи се появяват като вертикални и всички вертикални думи се появяват като хоризонтални. В действителност това е една и съща дъска, но погледнато от различен ъгъл.

Тъй като алгоритъмът работи по хоризонтален начин, генерирането на ходове от алгоритъма на база транспонираната дъска, което би ни дало хоризонтални ходове за транспонираната дъска, ще ни даде еквивалентните вертикални ходове за не транспонираната (оригиналната) дъска. Следователно ако подадем на алгоритъма едновременно оригиналната дъска и нейна транспонирана версия, ние накрая ще получим всички възможни хоризонтални и вертикални ходове за оригиналната дъска.

Важно е да се отбележи, че в „Скрабъл” хоризонталните думи се четат отляво надясно, а вертикалните думи се четат отгоре надолу. При транспониране на дъската, по същество, хоризонтални думи стават вертикални и обратното. Алгоритъмът работи въз основа на тези правила за четене на думите.

Следната фигура показва оригиналната дъска и нейната транспонирана версия.



*Фигура 27. Пример за дъска (отляво) и нейната транспонирана версия (вдясно). Хоризонталните думи отляво са станали вертикални вдясно, а вертикалните думи отляво са станали хоризонтален вдясно.*

**4.12.1.2. Котви**

Котвата е празно поле на дъската, което е в съседство с поле, върху което е поставена плочка. Условие за това, даден ход да е валиден е, че той трябва да бъде свързан с вече съществуваща дума на дъската. Тъй като алгоритъмът работи чрез добавяне плочки вляво и вдясно от дадени полета на дъската, той предварително трябва да знае кои полета са котви. Генерирането на ход, използвайки празни полета на дъската, които не са в съседство със заети полета, би довело до невалиден ход във всички случаи и би било ненужно използване на време. Като се предостави списък с котви на алгоритъма, той ще се опита да генерира думи, използвайки само тези котви. Трябва да се отбележи, че не всички думи, които са генерирани използвайки котва ще бъдат валидни, но никои от тях няма да нарушат посоченото по-горе условие по отношение на свързаността, което значително намалява времето за търсене на алгоритъма.

Следната фигура показва всички котви за даденото състояние на дъска.



*Фигура 28. Полетата, маркирани в зелено, са котвите за тази дъска, тъй като са съседни на вече запълнени полета на дъската вертикално или хоризонтално.*

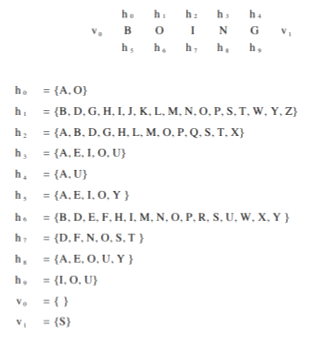
**4.12.1.3. Кръстосани проверки**

Една кръстосана проверка е набор от букви, които при изиграване върху дадено поле от дъската, имайки в предвид плочките, които се намират над и под това поле, която би била валидно разположение за тази плочка, биха довели до валиден ход. Тъй като алгоритъмът разширява дума по хоризонтален начин, той не взема в предвид дали някоя от буквите на генерираната дума участва в създаването на невалидна вертикална дума. Това се взима в предвид чрез използването на кръстосани проверки.

Правилата на „Скрабъл” гласят, че една дума може едновременно да използва само един ред и множество колони (хоризонтални думи) или множество редове и само една колона (вертикална дума). От това правило се разбира, че изиграването на една хоризонтална дума може да добави максимум една буква към колоните, които пресича и че вертикален ход може да добави максимум една буква към редовете, които пресича.

Празните полета на дъската, които не са вертикално свързани с други думи на дъската, нямат ограничение по отношение на кръстосаните проверки, т.е. всяка буква може да бъде поставена върху тези полета и това не би довело до вертикално невалиден ход. Обратното, празните полета на дъската, които са вертикално свързани с други думи на дъската, ще подлежат на ограничения от кръстосани проверки. Резултата от една кръстосана проверка ще бъде един празен списък или един списък, който съдържа набор от букви. Ако списъка е празен, това означава, че ако поставим буква, като част от хоризонтален ход, върху даденото поле на дъската, за което е била извършена кръстосаната проверка, това би довело до невалиден вертикален ход. Ако списъка съдържа букви, това означава, че ако поставим буква, като част от хоризонтален ход, върху даденото поле на дъската, за което е била извършена кръстосаната проверка, то тази буква трябва да бъде част от този списък, за да бъде хода валиден. Следователно, използвайки тези кръстосани проверки за отделните полета на дъската, алгоритъма може да намали своето време за работа, тъй като той ще избягва да използва букви върху полета, които не участват в списъците с букви, които са резултат от кръстосаните проверки.

Следната фигура е пример за списъците, получени в резултат от кръстосани проверки за набор от полета. Полетата отбелязани с „hx” се използват за хоризонтални ходове, а тези отбелязани като „vxЮ се използват за вертикални ходове. За полето „h0“, кръстосаната проверка дава списък „{A, O}”. Ако се играе хоризонтален ход, който ползва полето „h0“ (в посока от „h0“ до „h4“), единствените букви, които могат да бъдат поставени на полето „h0“, са „A“ и „O“, тъй като „AB“ и „OB“ са единствените валидни вертикални думи, които могат да се получат в резултат от валиден хоризонтален ход. За „v0“ кръстосана проверка дава списък „{}“, което означава, че не може да се направи валиден вертикален ход използвайки полето „v0“, тъй като нито една буква не може да бъде представка на думата „BOING“.



*Фигура 29. Пример за списъците, получени в резултат от кръстосани проверки за набор от полета.*

**4.12.2. Обобщение**

Имайки в предвид горните ограничения, първоначалното описание на алгоритъма се свежда до следното:

“*Алгоритъмът работи, като генерира всички възможни думи, използвайки всички букви на поставката на играча, всички котви на дъската, разширявайки всяка котва с комбинация от букви от поставката вляво и вдясно, стига буквите да са част от списъците, получени в резултат от извършените кръстосани проверки.”*

Важно е да се отбележи, че оригиналната дъска и транспонираната дъска ще имат различни координати за котвите и полетата от дъската, за които са извършени кръстосани проверки. Ето защо при инициализацията на един „MoveGenerator” обект, той трябва да бъде снабден с двете дъски, координатите за котвите на двете дъски и списъците от кръстосаните проверки на двете дъски. При инициализация трябва също да се подадат и поставката на играча, за който се генерира ход, „DAWG“ структурата, съставена от речника на играта, празен списък, който ще се актуализира с откритите от алгоритъма валидни ходове и ограничение във времето, при изтичането на което алгоритъма веднага спира. Метода за генериране на ходове „GetValidMoves()” на „MoveGenerator” обекта ще спазва споменатите ограничения при процеса на генериране.

**4.13. Генериране на ходове чрез „MoveGenerator“**

**4.13.1. „GetValidMoves()”**

„GetValidMoves()“ метода на „MoveGenerator“ обекта използва имплементация на методите от алгоритъма на Апел и Якобсън - „LeftPart()“ и „ExtendRight()“. На „GetValidMoves()” метода се подава един единствен параметър, който казва дали се генерират ходове за оригиналната или за транспонираната дъска. Оригиналната версия на алгоритъма на Апел и Якобсън разширява само едно поле вляво и вдясно, докато този метод извиква алгоритъма многократно, всеки път използвайки различна котва за разширение. Методът, също така, проверява дали е изминало зададеното ограничение във времето и прилага разбъркване в реда на подадените котви. По този начин, използвайки разбъркване и ограничение във времето, към метода се добавя и елемент на случайност и всяко ново негово повикване ще ни даде, като резултат, различен списък с валидни ходове. Това прави непредвидим начина, по който ще се развие играта.

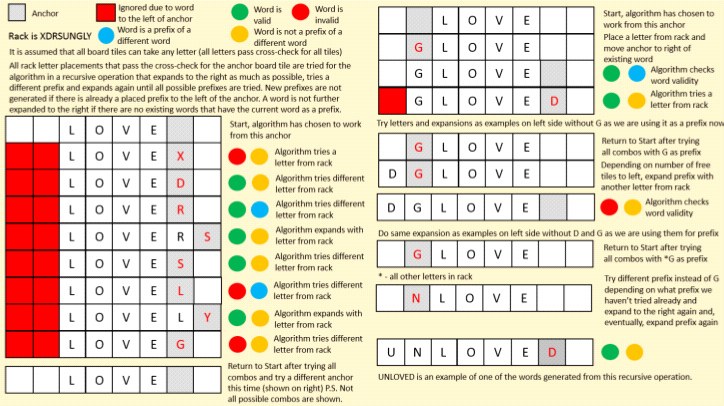
Алгоритъмът на Апел и Якобсън работи по рекурсивен начин. Следователно всяко разширение вляво или вдясно на дадено поле, от своя страна води до друго разширение вляво или вдясно, докато не се изпразни поставката на играча, докато не бъдат изпробвани всички възможни комбинации от букви в дадена посока или докато не бъде достигната границата на дъската.

Освен разширения в зависимост от поставката на играча и кръстосаните проверки за даденото поле, от което се правят разширения, друго условие, което се взема в предвид, е проверка дали буквата, избрана за разширение, е валидно продължение на думата, създадена до този момент от разширяването. Това става с помощта на „MatchPrefix()” метода на пакета, който ни предоставя „DAWG” структурата, който връща всички възможни продължения за дадена дума. Например, ако дъската има думата „CAT” и играча има на поставката си буквата „K“, която е валидна според кръстосаната проверка, „MatchPrefix()” ще ни каже дали съществуват думи, започващи с „CATK“. Ако такава дума не съществува, няма да се направи опит за разширяване с „K” и вместо това ще се направи опит със следващата буква в поставката на играча, отново използвайки „MatchPrefix()” метода. В крайна сметка ще се тестват всички букви от поставката на играча.

За всяка котва на дъската, „GetValidMoves()” метода проверява колко думата, която ще се генерира използвайки дадена котва, може да бъде разширена вляво (посредством „LeftPart()” метода). Ако има дума отляво на котвата, лимита на разширяване е 0. В противен случай лимита се инкрементира и се продължава разширението вляво, до достигане на съществуваща дума или границата на дъската. След този процес, ако лимита е 0, разширението вляво се прескача и се прави опит за разширение вдясно (посредством „ExtendRight()” метода). По време на отделните етапи от разширението, валидните възможни ходове се добавят към списък. Всеки ход има закачена към него информация и резултат въз основа на различни евристики, описани по-нататък в дипломната работа. Всяко разширение поставя плочки от поставката на играча на дъската и ги връща обратно в поставката, след като приключи с разширяването. След като за всички котви на дъската се извършат разширенията или след изтичане на зададения интервал от време, се връща списък с възможните ходове, който компютърният играч ще използва за да направи своя ход.

Следната фигура служи за визуална демонстрация на алгоритъма, използван за генерирането на ходове. Лявата половина на фигурата демонстрира „ExtendRight()“ метода, а дясната половина демонстрира „LeftPart()“ метода, който от своя страна многократно извиква „ExtendRight()“ метода. Логиката зад самите методи е разгледана по-подробно в следващите подточки.

Забележка: Трябва да се има в предвид, че методите се използват по рекурсивен начин, затова при прочита на следните подточки, е добра идея читателя да се позовава на предоставените фигури, тъй като те ще послужат за визуализация и по-лесно разбиране на това как работи алгоритъма.



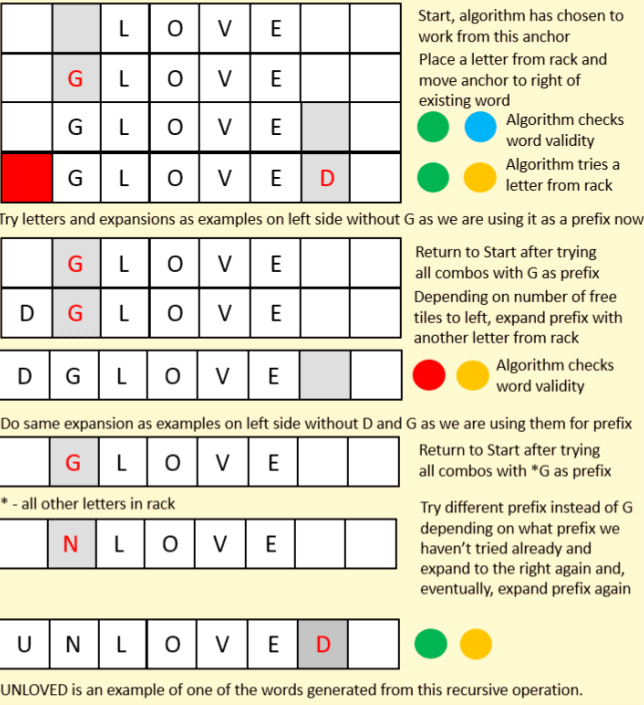
*Фигура 30. Визуална демонстрация на алгоритъма, използван за генерирането на ходове.*

**4.13.2. „LeftPart()”**

„LeftPart()” метода се занимава с разширения вляво от дадена котва. Ако е възможно разширение вляво, метода се вика като му се подава изградената досега лява част (изградена от вече съществуваща дума вляво от котвата или от новопоставени букви от предишни извиквания на „LeftPart()” метода). Използвайки подадената му лява част, методът извиква „ExtendRight()” метода, който от своя страна прави опити за допълнителни разширения вдясно от дадената котва. Това става като метода многократно извиква себе си („ExtendRight()” метода) по този начин създавайки списък от валидни ходове. В крайна сметка контрола върху приложението се подава на оригинално извикания „LeftPart()” метод, като той ще държи в себе си попълнен списък с валидните ходове, който ще се върне като резултат от извикването на метода и ще послужи в това, компютърния играч да направи своя ход.

Оригиналното извикания „LeftPart()” метод може да приключи по-рано, ако му е подаден интервал от време за работа. След изтичане на това време метода ще излезе от цикъла, в който се правят опити за разширения и ще се върне един съкратен списък с възможни ходове.

Следната фигура (дясната половина на оригиналната фигура) демонстрира „LeftPart()” метода.

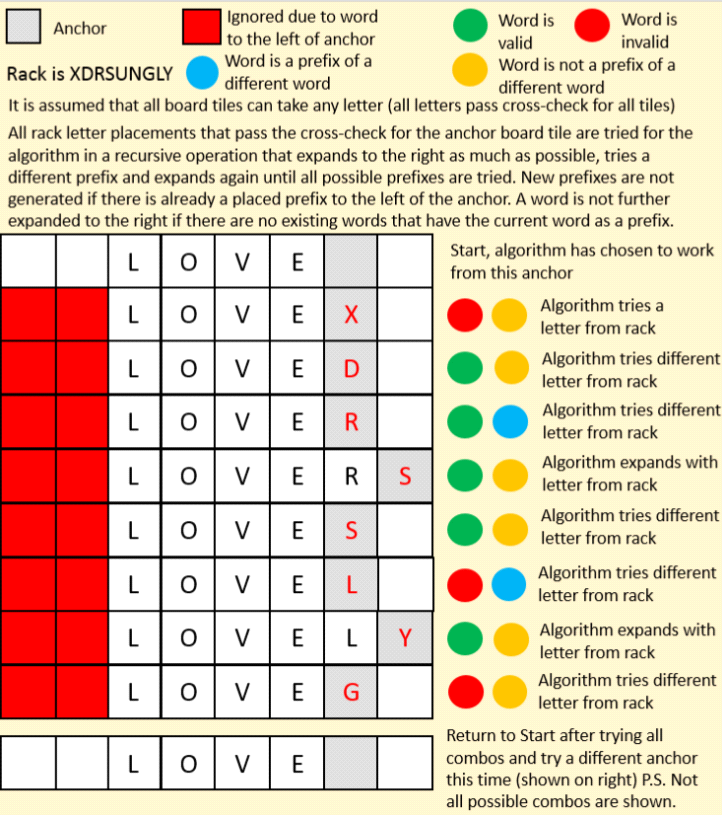


*Фигура 31. Демонстрация на „LeftPart()” метода, който от своя страна вика „ExtendRight()“ метода много пъти.*

**4.13.3. „ExtendRight()”**

„ExtendRight()” метода се занимава с разширения вляво от дадена котва. Той се вика от извикания преди това „LeftPart()” метод. Първото нещо, което „ExtendRight()” метода ще направи, е да провери дали думата, създадена до момента на извикването му, е валиден ход. Ако е, обект от тип „GeneratedMove” се добавя към списъка с валидни ходове. Този обект съдържа информация относно използваните букви, използваните полета на дъската, поставката, която му остава на играча, състоянието на дъската, преди да се направи опита за разширяване и др. След проверката на валидността на дума се прави опит за разширение вдясно. Ако вече има плочка на дъската, намираща се отдясно на съставената до момента дума, то тази плочка се използва в разширението. Следват множество самоизвиквания от „ExtendRight()” метода, които правят разширения вдясно използвайки останалите букви от поставката на играча и надграждат списъка с валидни ходове. В крайна сметка оригинално извикания „ExtendRight()” метод ще завърши и някакво време след това (след като приключат и извикванията на „LeftPart()” метода) ще бъде съставен списък с всички валидни ходове за дадена котва, стига да няма зададен интервал от време за работа.

Следната фигура (лявата половина на оригиналната фигура) демонстрира „ExtendRight()” метода.



*Фигура 32. Демонстрация на „ExtendRight()” метода.*

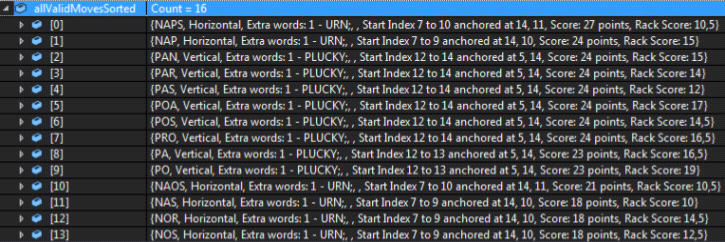
**4.13.4. „GeneratedMove“**

„GeneratedMove“ обекта съдържа информацията за валиден ход, който може да се играе на дъска, отнасяща се за началото и края на хода, използваните плочки от поставката на играча и полетата от дъската, които те заемат, самата дума и т.н. Един компютърен играч, избира хода, който да направи, на база пропъртитата на „GeneratedMove“ обекта. Компютърните играчи биват 2 вида - „High scorer” и „Rack balancer“.

Едно от пропъртитата на „GeneratedMove” обекта („Score“) държи резултата, с който ще се повиши крайния резултат на играча, ако извърши този ход. При изчисляването на резултата на пропъртито, се взимат предвид и първокласните полета, които хода ще покрие и дали изиграването на хода, потенциално, би довело до един или повече валидни вертикални хода, които се отбелязват като допълнителни думи за дадения ход. Всеки резултат от допълнителна дума се добавя към общия резултат. Един „High scorer” бот играч избира своя ход според това, на кой общия му резултат е най-добър.

„Rack balancer” бот играча се позовава на формула, получена от документа на Стивън Гордън. Резултатът от един хода зависи от баланса на гласни и съгласни на поставката след като се изиграе хода. Този баланс се изчислява чрез задаване на оценка на всяка буква на поставката и отрицателна оценка за нейните дубликати. След това се изчислява уравнение (посочено като „VCMix“ в документа) и резултатът от уравнението, се задава като стойност на пропъртито „RackScore“ на „GeneratedMove“ обекта. Ако тази стойност е висока, това означава, че поставката, с която ще остане играча след като изиграе хода, не е балансирана. Един „Rack balancer” бот играч избира хода белязан с най-ниска стойност на „RackScore” пропъртито.

Следната фигура показва списък с валидни ходове за дадено състояние на дъската, сортирани по най-висок резултат. Трябва да се има предвид, че това ход да има висок резултат не означава, че той има и балансирана поставка.



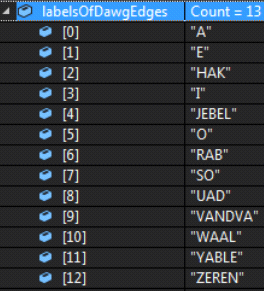
*Фигура 33. Списък валидни ходове за дадено състояние на дъската, сортирани по най-висок резултат.*

**4.13.5. „MatchPrefix()”**

„MatchPrefix()” e метод на „DawgSharp“ пакета, който ни предоставя „DAWG“ структурата. Чрез подаване на дума на този метод, той, потенциално, ще върне хиляди налични думи, които имат дадената дума като представка. За да филтрираме тези резултати, се използва „HashSet” структура, която използва компаратор, наречен „DawgEdgeEqualityComparer“, който посочва условията за уникалност и дублиране на елементите (думите) в структурата. Условието, което трябва да се изпълни от елемент, който ще бъде добавен към структурата е, че първата буква от този елемент не трябва да съществува като първа буква в някой от другите елементи в структурата. Всеки добавен елемент притежава префикса (дадената дума). Резултатът е съвкупност от думи, които могат да следват подадената на метода представка, като на всяка от тези думи е изрязана представката. Тъй като в структурата няма елементи, които започват с една и съща начална буква, размерът на резултата не може да бъде по-голям от броя на буквите в азбуката. Ако разгледаме дадена дума като звено от „DAWG” структура, първата буква на всеки елемент представлява път към различно звено, обозначено с тази буква. Следователно, ако някоя от буквите на поставката на играча не съвпада с началото на елемент от структурата, то не се прави опит за разширение вляво и вдясно с тази дума от „MoveGenerator” обекта.

Следната фигура показва списък от 13 думи с различни начални букви, чиято наставка е буквата „D“. Този списък казва на алгоритъма, че след буквата „D” има 13 възможни букви, които бижа довели до валиден ход. Поставянето на останалите 13 букви от азбуката, които не са в този списък, не биха довели до валиден ход и затова не се прави опит за разширение с тях.

Забележка: Думите „DA“, „DE“, „DHAK“, „DI“, „DJEBEL“ и т.н. са валидни думи от речника, използван от проекта.



*Фигура 34. Списък от 13 думи с различни начални букви, чиято наставка е буквата „D“.*

*.***4.13.6. Извършване на ход от компютъра**

След изпълнението на гореспоменатите методи системата ще има списък с ходове, които могат да се играят, като се има предвид текущото състояние на поставката на играча и дъската. Ако е ред на компютърният играч той ще използва този списък за да избере ход въз основа на това какъв тип играч е той („Rack balancer“, „High scorer“). Ако не може да се направи валиден ход (списъкът е празен), компютърът ще смени всички плочки на поставката си с нови от торбичката. Ако торбичката е празна, компютърът ще пропусне хода си.

**4.13.7. Празни плочки**

Ако празна плочка е в поставката на играча, при генериране на ходове от алгоритъма, когато той се опита да направи разширение вляво и вдясно използвайки тази плочка, тя ще се трансформира със всяка буква в азбуката и всяка от тези трансформации ще бъде използвана за разширение, което в последствие ще ни даде повече възможни ходове, но ще увеличи драстично времето за работа на алгоритъма. За да ги активирате, променете ред 650 във файла ScrabbleContext.

Забележка: Поради драстичното увеличение на използваните ресурси и времето за работа на алгоритъма, празните плочки няма да фигурират в демонстрацията.

**4.13.8. Gameplay**

Ако играта е била предадена на играч с хора, потребителят ще има (отключен) достъп до уеб страницата и ще може да играе дума, използвайки уеб страницата, като има предвид стойката, която е налична за текущия обект на играча. Ако играта бъде предадена на компютърен плейър, страницата ще се заключи, сървърът ще използва MoveGenerator обект, за да получи списък с ходове, ще се играе ход на дъската, базата данни ще бъде актуализирана и играта ще премине към следващия играч, който отново може да е човек или компютър.

Играта приключва при едно от следните условия:

* някой от играчите остане с празна поставка, след опит да я напълни с плочки от торбичката,
* някой от човешките играчи натиснете бутона „End Game” на уеб страницата.

След като приключи играта се изчисляват крайните резултати на играчите (добавя се или се премахва от крайния резултат според останалите в излишък плочки на поставките на играчите).

Отделно играта може да бъде рестартирана по всяко време с натискане на бутона „Reset Game” на уеб страницата. Това нулира резултатите на играчите, дава им нови поставки с плочки, дава нова пълна торбичка и изпразва дъската.



*Фигура 35. Пример за завършена игра. Торбичката е празна и играча „Dobromir” е изиграл всички плочки на поставката си.*

**5. Планиране и провеждане на тестове за верификация на предлаганото софтуерно решение. Представяне и оценка на получените резултати. Изводи.**

Този раздел ще тества всички компоненти на предложената имплементация на играта. Ще бъдат предоставени примери за готови игри, а функциите на играта ще се тестват индивидуално.

Имплементациите на техники и алгоритми вече са били изцяло тествани и дебъгнати по време на реализацията на проекта. Следователно тестовете в този раздел целят да потвърдят това, че гореспоменатите компоненти работят коректно. Също така ще се тестват компоненти, които не са разгледани подробно или не са споменати в дипломната работа, като например бутони в уеб страницата.

Следните резултати са от три различни игри на английски език.



*Фигура 36. Резултати от играта 1. Всяка дума в логовете на играчите („LIN“, „OI“, „LI” и т.н.) е потвърдена като налични в речника.*



*Фигура 37. Резултати от играта 1. Всяка дума в логовете на играчите („TIC“, „GOV“, „DIET” и т.н.) е потвърдена като налични в речника.*



*Фигура 38. Резултати от играта 1. Всяка дума в логовете на играчите („EA“, „OE“, „TIRE” и т.н.) е потвърдена като налични в речника.*



*Фигура 39. Резултат от игра 4 на български език. Всяка дума в логовете на играчите („ОМИЛ“, „МАК“, „СЕРАТ” и др.) е потвърдена като налична в речника.*

Следните функционалности са успешно тествани.

**5.1. Клиент**

* Изиграването на ход в началото на играта който не използва началното поле на дъската води до грешка „Invalid starting move” („Невалиден начален ход“).



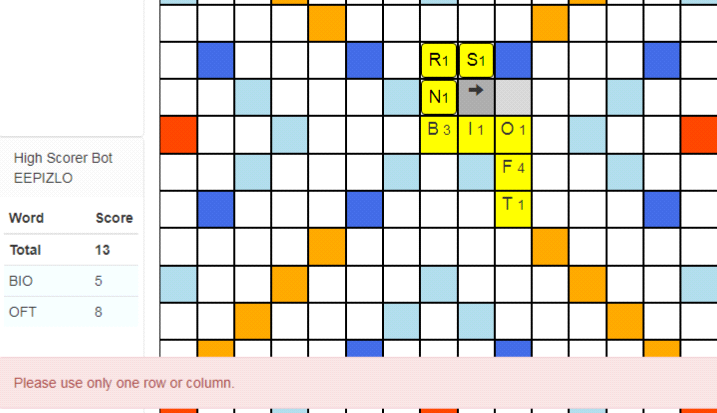
*Фигура 40. Тестване на валидиране от страна на клиента за използване на началното поле (началната плочка е розова).*

* Изиграването на ход, в който изиграните букви не са свързани води до грешка „Play not connected” („Несвързан ход“).



*Фигура 41. Тестване на валидиране от страна на клиента за свързаност на хода („R” и „N” са поставени).*

* Изиграването на ход, поставяйки букви, които са свързани както хоризонтално, така и вертикално води до грешка „Please use only one row or column” („Моля, използвайте само един ред или колона“).



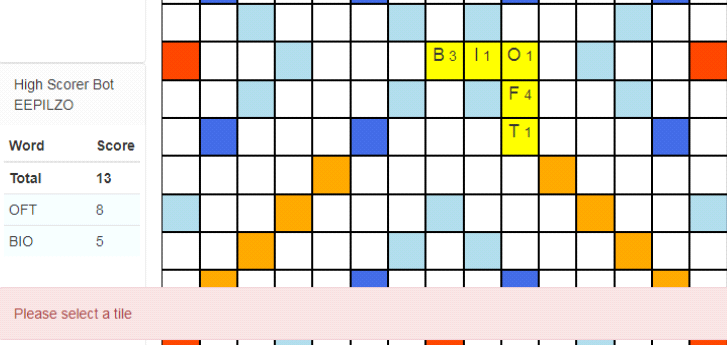
*Фигура 42. Тестване на валидиране от страна на клиента за това, дали ход е едновременно вертикален и хоризонтален („R“, „N” и „S” са поставени).*

* Изиграването на ход, който не е свързан със съществуваща дума на дъската води до грешка „Anchor not used” („Котва не е използвана“).



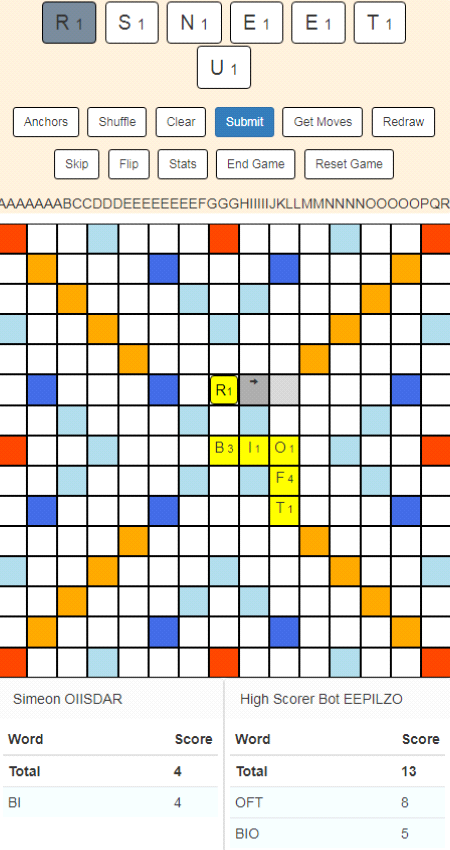
*Фигура 43. Тестване на валидиране от страна на клиента за използването на котва („R” е поставен).*

* Опит за поставяне на буква без да е избран поле на дъската води до грешка “Please select a tile” („Моля изберете плочка“).



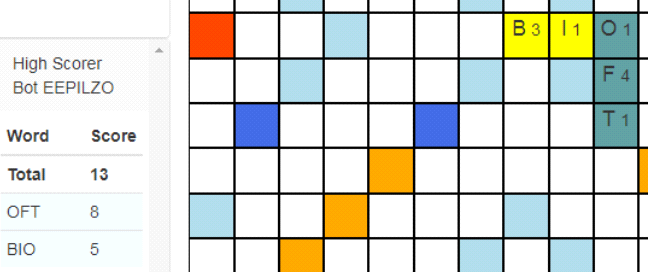
*Фигура 44. Тестване на валидиране от страна на клиента дали е избрано поле на дъската.*

* Натискане върху поле на дъската го прави активно и го оцветява в сиво със стрелка отгоре.
* Натискането на клавиша „Enter” симулира натискане на бутона „Submit“.
* Натискането на клавиша „Backspace” симулира натискане на бутона „Clear“.
* Натискането на клавиша „Space” променя посоката на стрелката на активното поле.
* Натискането на клавиш с буква, налична на поставката на играча, поставя буквата върху активното поле и маркира плочката като използвана.
* Активното поле се променя в зависимост от насоката на стрелката, когато се поставя буква.
* Когато активното поле се промени след поставянето на буква, следващата активно поле е най-близкото незаето поле. Ако е заеот, активното поле не се променя.
* След оразмеряване на прозореца дъската, поставката, контролите и логовете с думи на играчите се пречертават коректно (т.нар. „отзивчивост“ на страницата).



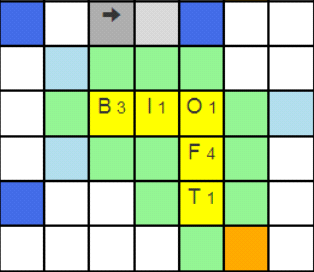
*Фигура 45. Тестване на отзивчивостта на страницата.*

* Височината на логовете на играчите съответства на височината на дъската.
* Задържане на курсора над дума от лога на играч я маркира в зелено на дъската.



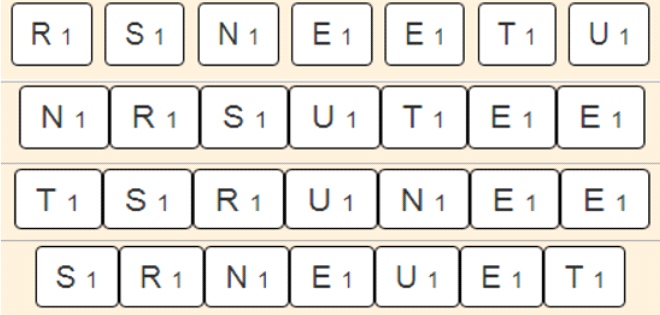
*Фигура 46. Тестване на маркирането на думи (курсора се намира над думата „OFT” от лога на играча).*

* Активното поле може да се смени чрез натискане върху друго поле или чрез използване на клавишите със стрелки на клавиатурата.
* Полетата на дъската, които вече имат буква върху тях, не могат да бъдат маркирани като активни.
* Натискането на бутона „Anchors” маркира всички котви на дъската в зелени.



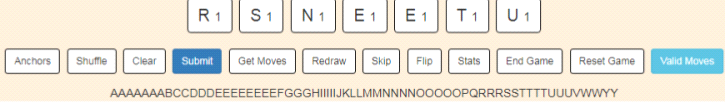
*Фигура 47. Тестване на маркирането на котви.*

* Натискане на бутона „Shuffle” разбърква поставката на играча.



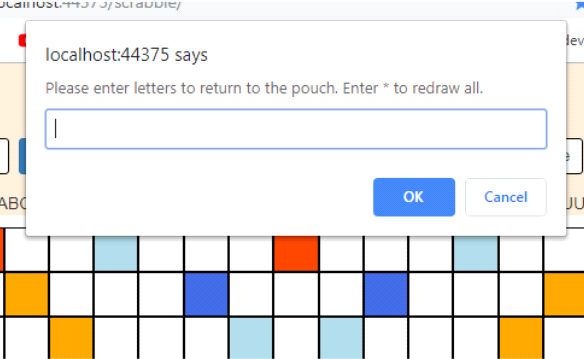
*Фигура 48. Тестване на функционалността на бутона „Shuffle“.*

* Натискане на бутона „Clear” връща всички поставени плочки, за текущия ход, обратно на поставката на играча.
* Натискане на бутона „Submit” извършва валидиране на хода от страна на клиента и изпраща данните на сървъра.
* Натискане на бутона „Get Moves” връща списък с възможни ходове от сървъра, който се съхранява в таблица, която се показва с натискане на бутона „Valid Moves” (този бутон става достъпен след натискане на „Get Moves“).

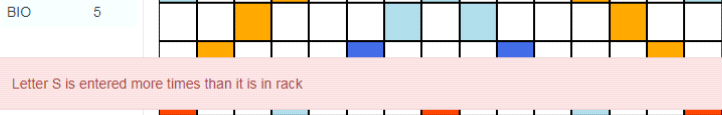


*Фигура 49. Тестване на това, дали бутона* „Valid Moves” се появява след натискане на бутона „Get Moves“*.*

* Натискане на бутона „Redraw” отваря прозорец, в който се въвеждат буквите, които играча иска да върне от поставката си към торбичката и за колкото букви той е върнал, толкова да изтегли. При въвеждане на „\*” се връщат всички букви от поставката на играча. Въвеждането на повече или различни букви от наличните на поставката води до грешка. След преизтеглянето на букви се дава ред на следващия играч.

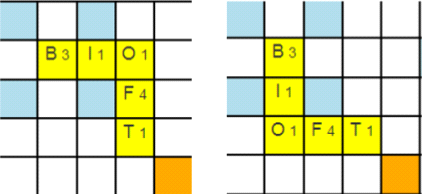


*Фигура 50. Тестване на появата на прозореца за въвеждане на букви, след натискане на бутона „Redraw“.*



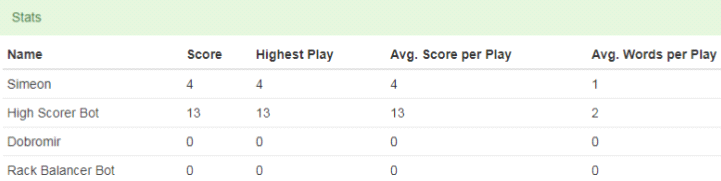
*Фигура 51. Тестване на прихващането на грешки при въвеждането на повече или различни букви от наличните на поставката (буквата „S” бе въведена два пъти, но налична само веднъж на поставката).*

* Натискане на бутона „Skip” дава ред на следващия играч.
* Натискане на бутона „Flip” връща транспонирана версия на дъската.



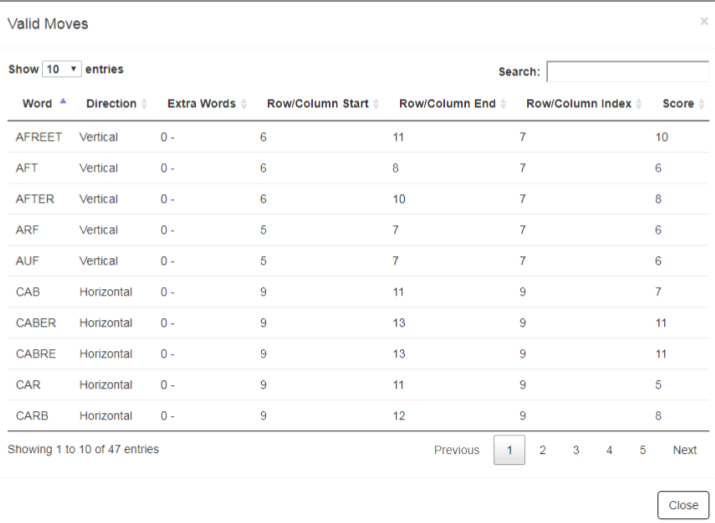
*Фигура 52. Тестване на транспонирането на дъската.*

* Натискане на бутона „Stats” показва или скрива таблица със статистика за играта.



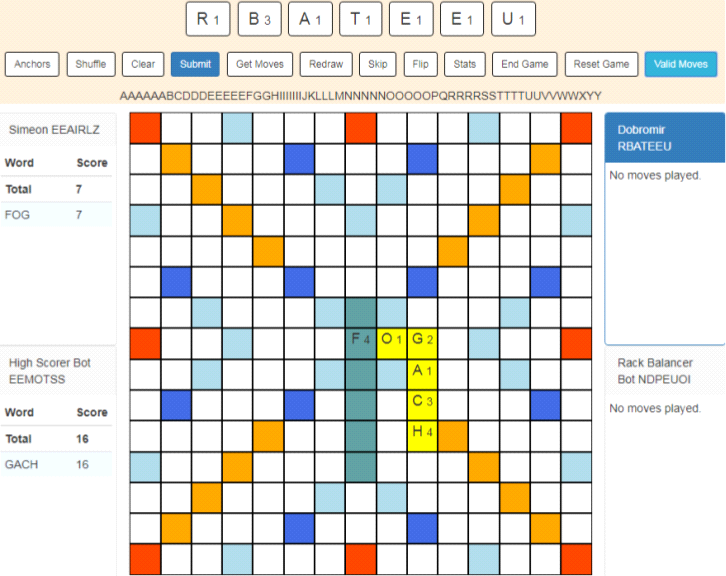
*Фигура 53. Тестване на визуализацията на статистиките.*

* Натискане на бутона „End Game” приключва играта, което заключва контролите и дъската и показва статистиките от играта.
* Натискане на бутона „Reset Game“ чисти дъската и логовете на играчите, възобновява торбичката и поставките на играчите.
* Натискане на бутона „Valid Moves” показва таблица със възможните ходове.



*Фигура 54. Тестване на визуализацията на таблицата с възможните ходове.*

* Натискането върху възможен ход от таблицата я затваря и маркира мястото на дъската, върху което ще се изиграе хода.



*Фигура 55. Тестване на избирането на възможен ход движения (избраната дума беше „AFREET“)*

* Отделните поставки на играчите са коректни.
* Съдържанието на торбичката, показано над дъската, е коректно.
* В лога с изиграните от играча думи, тези, които са с еднакъв цвят, са част от един и същи ход.

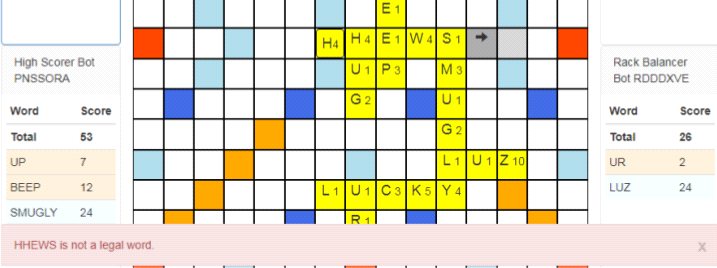


*Фигура 56. Тестване на оцветяването на думите. В лога на играча „Dobromir“, думите „OE“ и „EA“ са част от един и същи ход, докато думата „RE“ беше единствената изиграна дума от последния му ход.*

**5.2. Сървър**

Следните точки са проверки, които се извършват на сървъра. Проверките са тествани и са коректни:

* Изиграването на неправилна дума води до грешка от сървъра.



*Фигура 57: Тестване на валидацията на думи от страна на сървъра.*

* Изиграната от играч дума е валидна дума от речника.
* Изиграната от играч дума използва котва от дъската.
* Изиграната от играч дума спазва кръстосаните проверки.
* Състоянието на дъската се възобновява.
* Използването на букви от поставката ги премахва от нея.
* Колкото букви от поставката са използвани при извършването на ход, толкова плочки се теглят от торбичката.
* Плочките, изтеглени от торбичката, са произволни.
* Тегленето на плочки от торбичката ги премахва от нея.
* Дъската, която е показана на потребителя, отговаря на дъската, която се съхранява.
* Резултата, при изиграване на дума, се изчислява правилно.
* Бонусите, от първокласните полета, се изчисляват правилно.
* Резултата от ход, в резултат на който са съставени няколко думи, се изчислява коректно.
* Бонусите, от първокласните полета, се премахват след като полето е било използвано.
* Списъка с котви е правилен.
* Набора от кръстосани проверки е правилен.
* Генерираните ходове са правилни.
* Промяна в състоянието на играта се отразява в базата.
* Съставения речник е правилен.
* Компютърния играч си преизтегля поставката ако не може да направи валиден ход и торбичката не е празна.
* Играта приключва ако на някой играч поставката е празна.
* Резултата на играчите, които в края на играта имат плочки на поставката, се санкционира, а този на играча, който е успял да изиграе всичките си плочки, се възнаграждава.

**5.3. Оценка на получените резултати**

В този раздел са тествани по-голямата част от функционалностите от страна на клиента и на сървъра на приложението. От направените тестове следва, че разработеното „Скрабъл” приложение е стабилно и годно за игра. Проектът беше оценен от мен и моите приятели Симеон и Добромир, с които заедно направехме тестови игри на приложението. Откъм функционалности играта не прави грешки и работи правилно.

По отношение на първоначалните цели на проекта, повечето от тях са изпълнени успешно. Единствените, които не са били реализирани, са компютърния играч да извършва симулации, за да избере по-добри ходове, и да променя стратегията си през различни фази на играта.

По отношение на поставения в началото въпрос - може ли играта да реализирана с нови технологии, позовавайки се на наличните проучвания, имплементации и техники и може ли да се подобри – проекта е успешен. „C#“ и „.NET“, в момента на тяхното използване, са сравнително модерни инструменти и с тяхна помощ успешно бе имплементирана играта „Скрабъл“ като „single-page“ уеб приложение с допълнителни функционалности като игра с компютърен опонент и многоезичност.

**6. Заключение**

Въпреки че не е перфектен, проектът е успешен, тъй като всички основни цели са постигнати. Всяка от основните точки в този документ играе ключова роля при писането на този дипломен проект. Първа точка (Постановка на проблема. Цели и задачи. Мотивация) представи на читателя обобщение на проекта, целите на проекта и намерението зад всички следващи точки. Втора точка (Теоретичен преглед на сходни с целта на проекта решения, технологии и средства за имплементация) представи историята на играта, алгоритмите и реализациите, използвани в миналото. Трета точка (Проект на предлаганото софтуерно решение) въведе технологиите, използвани в реализацията. Четвърта точка (Детайлно описание на предлаганата софтуерна реализация) демонстрира как са били използвани технологиите и как те работят заедно. Пета точка (Планиране и провеждане на тестове за верификация на предлаганото софтуерно решение. Представяне и оценка на получените резултати. Изводи) тества всички функционалности на крайното приложение, за да потвърди, че то е напълно работоспособно.

Изграждането на този проект от нулата се оказа една доста трудна задача, но крайният резултат, въпреки неговите недостатъци, е едно доста прилично уеб приложение и голям личен успех. Отправям благодарности към моя ръководител и колега Пламен Тодоров (ръководител екип програмисти в Abrites Ltd.), без който проектът не би бил реализиран до тази степен, както и на Ива Николова, комуникацията с която ми разясни много неясноти покрай дипломната работа като цяло.

**7. Използвана литература**

Теоретичният преглед съдържа информация от публикувани документи, посочени в дипломната работа. При самото разработване на проекта бяха използвани предимно онлайн ръководства и уроци. Следните линкове помогнаха при разработването на проекта и са използвани до такава степен, че би било трудно да разпише.

<https://docs.microsoft.com/> - документация от „Microsoft“, която значително помогна при използването им технологии.

<https://stackoverflow.com/> - общност от програмисти, които предоставят отговори на въпроси свързани с програмирането.

Списания, документи и полезни връзки са поставени по-долу:

1. Janakshi Dulanga D., An Artificial Intelligence Based Scrabble Game for Sinhala Language,Proceedings of APIIT Business, Law & Technology Conference, 2017 Colombo, Sri Lanka. ISBN978-955-7678-02-3
2. APPEL, A. and JACOBSON, G. (1988). The World’s Fastest Scrabble Program. Communications of the ACM, 31(5).
3. Sheppard, B. (2002). World-championship-caliber Scrabble ☆☆SCRABBLE® is a registered trademark. All intellectual property rights in and to the game are owned in the USA by Hasbro Inc., in Canada by Hasbro Canada Corporation, and throughout the rest of the world by J.W. Spear & Sons Limited of Maidenhead, Berkshire, England, a subsidiary of Mattel Inc. Artificial Intelligence , 134(1-2), pp.241-275.
4. Rajkumar, P. (n.d.). A Survey of Monte-Carlo Techniques in Games. Master’s Scholarly Paper.
5. Gordon, S. (1993). A COMPARISON BETWEEN PROBABILISTIC SEARCH AND WEIGHTED HEURISTICS IN A GAME WITH INCOMPLETE INFORMATION. Department of Mathematics. East Carolina University.
6. Connolly, F. and Gren, D. (2012). Smart Scrabble playing - strategies and their impact. DD143X. Kungliga Tekniska Högskolan.
7. Gordon, S. (1994). A faster scrabble move generation algorithm. Software: Practice and Experience, 24(2), pp.219-232.
8. Fatsis, S. (2014). Slate’s Use of Your Data. [online] Slate Magazine. Available at: <https://slate.com/human-interest/2014/09/scrabble-copyright-dispute-hasbro-says-it>- owns-the-scrabble-dictionary-players-beg-to-differ.html [Accessed 7 Oct. 2018].
9. Hasbro. (n.d.). Scrabble Rules | Official Word Game Rules | Board Games . [online] Available at: <https://scrabble.hasbro.com/en-us/rules> [Accessed 31 Oct. 2018].
10. Wikipedia. (2014).GADDAG. [online] Available at:https://en.wikipedia.org/wiki/GADDAG [Accessed 27 Oct. 2018].
11. Wikipedia. (n.d.). Scrabble. [online] Available at: <https://en.wikipedia.org/wiki/Scrabble> [Accessed 27 Oct. 2018].
12. Srivastava, B. (2017). History оf C# Programming Language. [online] C-sharpcorner. Available at: <https://www.c-sharpcorner.com/blogs/history-of-c-sharp-programming>- language [Accessed 1 Apr. 2019].
13. Purdy, K. (2011). What Is the .NET Framework, and Why Do I Need It? . [online] Lifehacker. Available at: <https://lifehacker.com/what-is-the-net-framework-and-why-do-i-need-it>- 5791578 [Accessed 1 Apr. 2019].
14. Roth, D., Anderson, R. and Luttin, S. (2019). Introduction to ASP.NET Core . [online] Microsoft. Available at: <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore>- 2.2 [Accessed 2 Apr. 2019].
15. Atwood, J. (2008). Understanding Model-View-Controller. [online] CodingHorror. Available at: <https://blog.codinghorror.com/understanding-model-view-controller/> [Accessed 2 Apr. 2019].
16. MDN Web Docs. (n.d.). Client-Server Overview. [online] Available at: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client> Server\_overview [Accessed 2 Apr. 2019].
17. Miller, R., Vega, D., Dykstra, T., Wenzel, M. and Lambson, B. (2016). Overview - EF Core [online] Microsoft. Available at: <https://docs.microsoft.com/en-us/ef/core/> [Accessed 2 Apr. 2019].
18. Hui, K. (2018). Developing Web Apps. [online] Campusmoodle. Available at:http://campusmoodle.rgu.ac.uk/mod/resource/view.php?id=3135893 [Accessed 3 Apr. 2019].
19. Gibb, R. (2017). What is Lazy Loading? | StackPath Blog. [online] StackPath. Available at: <https://blog.stackpath.com/glossary/lazy-loading/> [Accessed 4 Apr. 2019].
20. bzaar (2018). DawgSharp. [online] GitHub. Available at: <https://github.com/bzaar/DawgSharp> [Accessed 4 Apr. 2019].
21. Chortle.ccsu.edu. (n.d.). Transpose. [online] Available at: <https://chortle.ccsu.edu/VectorLessons/vmch13/vmch13_14.html> [Accessed 4 Apr. 2019].