Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Проекти

курс Структури от данни и програмиране
за специалност Информатика
зимен семестър 2018/19 г.

1: Синхронизиране на директории

В рамките на този проект трябва да разработите приложение, което сравнява и синхронизира две директории.

Приложението ви трябва да получи като вход от потребителя път до две директории. След това то трябва да ги сканира, да анализира получените резултати и да предложи на потребителя да ги уеднакви, като го информира какви операции ще бъдат извършени.

Приложението ви трябва да получава входните си параметри от командния ред. Общият формат е:

```
sync.exe <command> [<args>]
```

Първият параметър, `<command>`, указва каква операция да се извърши. След него може да се подадат нула, един или повече аргументи.

Командата за обхождане и анализиране на двете подадени директории е `analyze`:

```
sync.exe analyze [<args>] <left> <right>
```

Последните два аргумента указват кои са входните директории. Първата подадена директория ще наричаме "лява", а втората – "дясна". По-долу в текста за по-лесно ще ги бележим с L (от "Left") и R (от Right).

За улеснение, ще считаме, че в двете директории няма "връзки" (soft links, hard links, junctions) или специални файлове. Може да считате, че съдържанието на всяка от двете директории образува дърво. В тях не може да има цикли, нито възли с повече от един родител.

Възможно е в директориите да има файлове, до които приложението ви не може да получи достъп. Например може да има файл, който е отворен в друго приложение; файл, за който нямате права за достъп и т.н. Приложението ви трябва да може да открива кога няма достъп до даден файл или директория и да предприема подходящи действия (например може да пропуснете такива файлове/директории, а в края на работата да изведете съобщение, в което да укажете какво не е било обработено).

Командата получава аргумент, който указва какъв вид синхронизация искаме да направим:

- `mirror` – R трябва да стане точно копие на L.
- `safe` – Никакви файлове не трябва да се изтриват или променят. Ако даден файл липсва в една от двете директории, но е наличен в другата, той се копира в съответната посока.
- `standard` – Никакви файлове не трябва да се изтриват. Файловете, които съществуват в една от директориите и липсват в другата, се копират в съответната посока. Ако даден файл съществува и в двете, но е с различно съдържание, трябва да запазите това копие, което е с по-нова дата. Другото копие трябва да се презапише с тази, по-нова версия.

По-долу е даден пример за стартиране на програмата:

```
sync.exe analyze standard C:\Temp C:\Directory\SubDirectory
```

За да обходите двете подадени директории, използвайте итератора `recursive_directory_iterator` от въведената в C++17 библиотека Filesystem. За повече информация вижте например:
https://carlosvin.github.io/posts/recursive-directory-iterator/
https://www.bfilipek.com/2017/08/cpp17-details-filesystem.html
https://en.cppreference.com/w/cpp/filesystem/recursive_directory_iterator
https://docs.microsoft.com/en-us/cpp/standard-library/recursive-directory-iterator-class?view=vs-2017

При обхождането трябва да запазите информация за всеки от намерените файлове (например размер, дата на последна промяна, хеш на съдържанието и т.н.). След това трябва да сравните информацията за двете дървета и да откриете какви са разликите между тях. Например:

- Възможно е да има файлове и/или директории, които се намират в едното дърво, но не и в другото;
- Възможно е да има файлове, които се намират и в двете дървета, но не съвпадат по своите дата, размер, съдържание;
- и т.н.

За да може да извършвате сравнения на съдържанието на различните файлове, използвайте подходящ алгоритъм за хеширане и работете с генерираните хешове, вместо да сравнявате всички файлове байт по байт. Тъй като е възможно да се получи колизия (два различни файла да имат еднакви хешове), потребителят трябва да може да укаже какво ниво на сигурност очаква. Ако на операцията `analyze` се подаде аргумент `hash-only`, файловете се сравняват само и единствено използвайки хешове. В противен случай, ако имаме колизия и всички други свойства на файловете съвпадат (еднакъв размер, дата и т.н.), програмата все пак трябва да ги сравни байт по байт, за да гарантира, че те наистина са еднакви.

Програмата трябва да може да открива преименувани и/или преместени файлове. Това са файлове, които се намират и в двете дървета, но с различни имена и/или относителни пътища спрямо корена. За такива файлове ще бъде по-бързо, вместо да се копират, да се преименуват и/или преместят. По този начин, в зависимост от входните директории, операцията "огледало" (mirror) може да се ускори значително.

След като приложението ви приключи с анализа и открие разликите между двете директории, то трябва да даде информация на потребителя за това какви стъпки са необходими, за може те да се синхронизират. За целта трябва да се създаде текстов файл, в който са описани всички нужни операции. Възможни операции са например:

- копиране на файл от едната към другата директория;
- изтриване на файл;
- преименуване на файл;
- преместване на файл;
- и други.

По-долу е даден пример за това как би могъл да изглежда изходният файл с операции за дадено сканиране. Първите два реда в него указват кои са директориите, които се обработват. Всеки ред указва една операция, която трябва да се извърши. В отделните редове, всеки път започва с L или R. L указва лявата директория, а R – дясната. (Ако предпочитате, използвайте пълните думи LEFT и RIGHT вместо L и R) Символът # указва началото на коментар – текстът след него се игнорира. Можете да използвате коментари, за да поясните отделните операции. Можете да промените или разширите формата на файла, ако решите, че е необходимо.

```
LEFT is C:\Temp
RIGHT is C:\Directory\SubDirectory
COPY L\1.txt R\1.txt   # Left copy is newer
CREATE-DIR R\Z   # Missing on the right
COPY L\Z\2.txt R\Z\2.txt   # Missing on the right
DELETE R\SomeDir\3.txt
DELETE R\SomeDir
```

След като генерира файла с операции, програмата прекратява своето действие. След това потребителят може да го прегледа и да прецени дали иска да изпълни операциите или не. Възможно е също така да го редактира, като промени или изтрие отделните редове. Така например, ако той реши, че не иска да копира файловете от поддиректорията Z и освен това иска да запази файла 3.txt, потребителят може да промени файла по следния начин:

```
LEFT is C:\Temp
RIGHT is C:\Directory\SubDirectory
COPY L\1.txt R\1.txt # Left copy is newer
CREATE-DIR L\SomeDir
COPY R\SomeDir\3.txt L\SomeDir\3.txt
```

Забележете, че редът на операциите има значение. Например, за да копираме файла 3.txt, най-напред трябва да създадем директорията SomeDir. Ако разменим редът на тези операции или пропуснем създаването на директория, синхронизацията няма да бъде успешна.

След като приключи с преглеждането на файла, потребителят може да изпълни така зададените операции. За целта се използва командата `perform`. Като входен параметър, тя получава път до файл съдържащ списък с операции. Например:

```
synchronize.exe perform sync.txt
```

За извършване на операциите, използвайте функциите от библиотеката Filesystem от C++17 (например copy, remove и т.н.). За повече информация вижте например: https://en.cppreference.com/w/cpp/filesystem

Програмата трябва да предлага възможност за частично (поблоково) копиране. Например възможно е даден голям файл да се намира и в двете дървета, но съдържанието му да е различно. Понякога за такива файлове промените са само в малка част от съдържанието. Например един файл с размер 4 гигабайта може да е бил променен само в своя край. В такъв случай ще бъде удачно вместо да се копира целият файл, да се копират само различните части. Когато пресмятате хешовете за такива файлове, вместо да пресметнете един хеш за целия файл, може да бъде по-удачно да го разделите на

блокове с еднакви размери (например по 64 мегабайта), да пресметнете техните хешове и след това да представите файла като наредена последователност от хешове. При синхронизацията на такива файлове трябва да се копират само блоковете, които имат различни хешове. За да укаже, че иска да използва тази опция, потребителят добавя аргумент `block` при анализа на две директории. Например:

```
sync.exe analyze standard block C:\Temp C:\Directory\SubDirectory
```

Сами изберете как да реализирате тази опция. Като минимум тя трябва да се отчете:

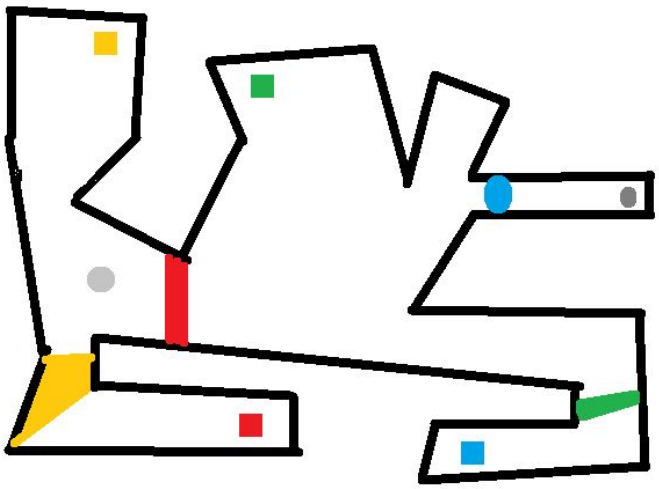
- По време на анализа и сравнението между двете директории;
- За файловете, за които тя е приложима може да се наложи да въведете нова, специална команда във файла с операции;
- При изпълнението на операциите ще трябва да може да изпълните частично копиране.

Количеството памет, което вашата програма използва не трябва да се влияе от обема на файловете, а само от техния брой. Пространствената сложност трябва да бъде от порядъка на O(N), където N е броят на файловете в двете дървета. Тя трябва да бъде O(1) спрямо размера на файловете.

Възможно е между изпълнението на двете фази, съдържанието на двете директории да се промени. Например правата за достъп до даден файл може да се променят, потребителят може да изтрие или премести даден файл и т.н. Сами преценете как да адресирате този проблем.

2: Лабиринт

Карта на лабиринт ще представяме чрез изображение в Bitmap формат. Например:



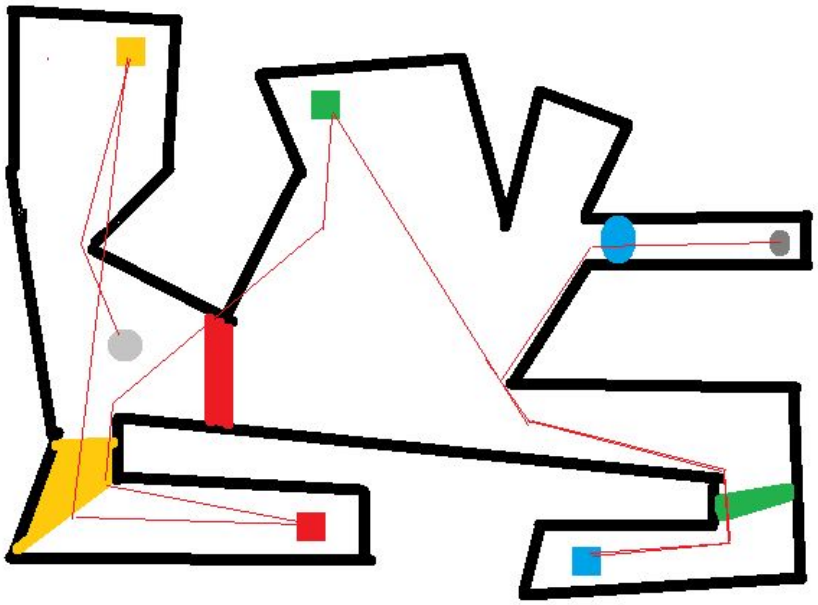
В изображението всеки пиксел указва различен обект върху картата. Черните пиксели (0,0,0) ще представят непроходими зони (например стени). Белите пиксели (255,255,255) представят проходимите пространства. Началната зона, от която тръгва обхождането е отбелязана със светло сив (195,195,195) цвят, а изходната – с тъмно сив (127,127,127). Възможно е да има повече от една изходна зона.

В лабиринта е възможно да има зони оцветени с цветове различни от бяло, черно и сиво. Това са зони, през които може да се мине само ако разполагаме със съответния ключ. Ключовете отбелязваме като квадрати с фиксиран размер (20 на 20 пиксела) и със същия цвят като на вратите. За улеснение считаме, че никоя от специалните зони не е квадрат с размер точно 20 на 20 пиксела.

Ключовете не се губят при преминаване през специалните зона. Например ако в лабиринта има няколко жълти зони, можем да преминаваме през всяка от тях, стига да разполагаме с жълт ключ.

Освен ключовете, всички други части на лабиринта могат да бъдат с произволен размер и форма. Например стените могат да са с различна дебелина, заоблени, под произволен ъгъл; специалните зони могат да бъдат като дъга, кръг, правоъгълник, с неправилна форма и т.н.

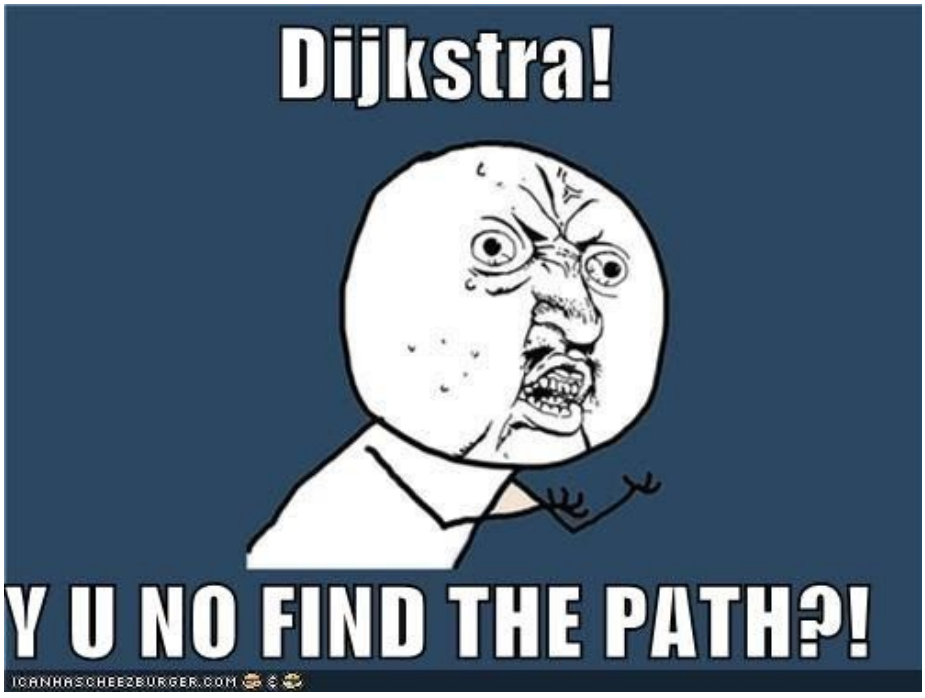
Вашето приложение трябва да зареди лабиринт от дадения вид и да генерира ново изображение, в което е показано как може да се излезе от лабиринта. Например:



Освен изображението, трябва да се генерира и текстов файл, в който да се опише намереният път. Пътят описваме като редица от точки P_1, P_2, … P_N, през които трябва да се премине последователно, за да се стигне до изхода. Придвижването между две последователни точки става по права линия. Всеки ред от файла трябва да съдържа координатите на една точка от пътя – първият ред съдържа P_1, вторият – P_2 и т.н. По-долу е даден пример за съдържанието на такъв файл.

```
10 20
500 90
1 300
```

Сами изберете какво да направи програмата, ако не съществува начин да се излезе от лабиринта. Например може да изведете картинка на намръщено smiley, да изведете лабиринта зачеркнат без да нарисувате път в него, да изведете текст "no solution" в текстовия файл и т.н.

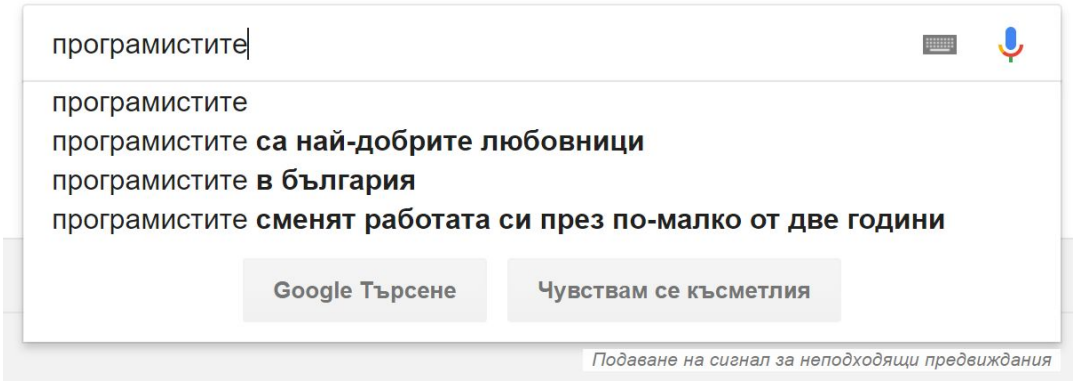


Фигура 2.1: Възможно е работата по проекта да ви отведе по неведоми пътища

3: Autocomplete

В рамките на този проект трябва да разработите модул за autocomplete. В него ще можете сами да изберете алгоритъма, с който да работите.

Модулът трябва да предоставя функционалност подобна на тази, която позволява на Google Search да визуализира списък с предложения, когато започнете да пишете дадена фраза:



Фигура 3.1: Google винаги дава вярна информация

За разлика от инженерите от Google, Вие ще трябва да решите много по-проста задача. Като вход от потребителя ще получите път до текстов файл. Този файл ще съдържа *N* на брой символни низа. Това ще бъдат различни думи и/или фрази, по една на всеки ред. Вие трябва да заредите съдържанието на файла в подходящ контейнер. След това трябва да можете, по префикс *P*, съдържащ *L(P)* букви, да връщате списък от не повече от *K* предложения за това как може да се довърши префикса.

В рамките на проекта трябва да решите следните задачи:

Задача 1. Потърсете в Интернет информация за това какви подходи съществуват за решаване на тази задача. Започнете проучването си от източниците дадени в края на това описание.

Задача 2. Изберете подходяща структура от данни, с която да решите задачата. Вашето решение трябва да може да работи достатъчно бързо, така че потребителят да може да получава предложения в реално време, докато пише на клавиатурата. Също така, броят на входните символни низове *N* може да бъде много голям.

Помислете за това дали и какви ограничения за *P* да поставите. Например може да поискате *L(P)* да бъде поне 3 или повече букви, преди да започнете да давате предложения.

Относно *K*: очевидно, ако потребителят въведе префикс, който не съвпада с никой от низовете в контейнера, броят на предложенията ще бъде 0. Ако обаче има поне едно съвпадение, трябва да се помисли за това дали да се постави ограничение. Ако например има 1 000 000 съвпадения, няма как да предложите всички на потребителя. Затова

потребителят трябва да може да указва някаква стойност за *K*, например 10, след което вашето решение трябва да предлага най-много *K* символни низа при дадено търсене. *K* трябва да може да се променя динамично по време на изпълнение. Например ако потребителят реши, че 10 низа не са му достатъчни, той може да увеличи *K* до 20.

Обърнете внимание и на обема памет, който ще изисква избраното от вас представяне. За целите на проекта приемаме, че входната азбука не се ограничава само до латинските букви и може да включва произволни Unicode символи. Затова и приложението Ви трябва да работи с Unicode, а не с еднобайтови `char` символи.

След стъпка 1 и 2, още преди да започнете да програмирате решението си, трябва да съгласувате това, което сте избрали като подход за решаване на задачата, с екипа на курса. Така ще се избегне ситуация, в която сте избрали прекалено проста или прекалено сложна за рамките на курса СД. За целта изгответе кратко описание (достатъчно е половин до една страница), в което:

- опишете накратко коя структура от данни сте избрали (може да са няколко);
- опишете кои помощни алгоритми ще използвате (ако има такива);
- кажете каква ще бъде архитектурата на решението (достатъчно е кратко описание или диаграма);
- обосновете взетите от вас решения.

Ако се затруднявате, прочетете как се правят дизайн документи, например в тази статия.

Качете описанието, например като MS-Word или PDF документ, във формата за предаване на проект и се свържете с екипа. Ще получите от нас потвърждение или забележки за неща, които трябва да се променят. Преминете към задача 3 едва след като бъдат изчистени всички забележки свързани с избрания от вас подход.

Задача 3. Реализирайте избрания от вас алгоритъм. Като минимум трябва да реализирате клас Autocomplete, който да предлага поне следните операции:

- `Insert` – добавя нов символен низ в контейнера;
- `Suggest` – получава като аргумент един символен низ – префикс, по който трябва да се направи търсете. Операцията да връща нула, един или повече символни низа – предложения за това как да се довърши префикса.
- Трябва да има възможност за извличане и промяна на стойността на K.

Задача 4. Тествайте решението си, като му подадете голям обем входни данни и го поставите под натоварване, за да проверите колко бързо то може да обработи голям брой заявки.

Задача 5. Напишете кратка програма, която демонстрира работата на решението. Програмата трябва да може да зареди списък от символни низове от файл. След това тя трябва да позволява на потребителя да въвежда различни символни низове, да прави по тях търсене и да извежда предложенията на екрана. Програмата трябва да позволява на потребителя да променя стойността на *K*.

Ако имате възможност, реализирайте програмата така, че да има графичен интерфейс. Изобразете поле за въвеждане (edit box) и под или над него, в реално време, извеждайте предложенията, които решението ви генерира.

Anti Ajanki, Data structures for fast autocomplete
https://www.futurice.com/blog/data-structures-for-fast-autocomplete/

Algorithm for autocomplete?
https://stackoverflow.com/questions/2901831/algorithm-for-autocomplete

4: Digger

В рамките на този проект трябва да реализирате играта Digger. Можете да играете нейна онлайн версия тук.

Правила на играта

В играта потребителят контролира копача, който може да копае вертикални или хоризонтални тунели.

На всяко ниво има поне един вече прокопан тунел и копачът се намира на него в началото на играта.

Играчът има фиксиран брой стартови животи.

Периодично, в горната-дясна страна на нивото, върху прокопана част от тунел се генерират врагове.

За всяко ниво има фиксиран брой врагове.

Враговете постоянно вървят към копача и ако го докопат, той губи живот и нивото стартира наново, като целият прогрес до сега бива запазен.

Също така, на картата са разпръснати смарагди и торби със злато.

Смарагдите могат да бъдат изкопани, когато играча мине през тях, като това му носи фиксиран брой точки.

Торбите със злато от своя страна могат да бъдат местени наляво и надясно. Ако под тях бъде прокопан тунел, торбата се активира и след няколко секунди пада. Ако докато бива бутана, дадена торба се намери над празно пространство, то тя пада веднага, без пауза. Когато златна торба започне да пада, тя смазва всичко живо под нея, като това включва както чудовищата, така и играча, който губи живот ако бъде сполетян от тази злощастна съдба. Ако една торба пропадне повече от един ред надолу, при удрянето ѝ в земята, тя се разбива и генерира злато, което играчът може да събере. В противен случай, торбата остава цяла.

Когато играчът събере златото от една торба, той получава възможност да стреля. В такъв случай, в посоката, в която гледа копачът, се изстрелва снаряд, който лети по права линия. Ако снарядът уцели чудовище, то умира. В противен случай, ако уцели земя или излезе от картата, снарядът се губи.

За пълните правила може да прочетете wikipedia страницата на играта или да я поиграете сами.

Имплементация

Вашето приложение може или да рисува върху конзолата, като за целта може да използвате Windows API, или да използвате някоя библиотека за rendering като SDL2.

Вашата цел е да направите така, че в играта могат да се случват всички действия. Това включва няколко неща които се случват едновременно. Например, на един кадър може да се генерира чудовище, всички други врагове да се придвижат на съседна позиция, торба със злато да пада, друга торба да се активира, смарагд да бъде изкопан и естествено копача да се придвижи. За тази цел най-вероятно ще трябва да имплементирате някакъв вид game loop. Ще трябва да имплементирате и начин потребителя да контролира копача, като това не блокира играта (например дори и да не предприемате никакви действия, чудовищата трябва да се придвижват, когато копачът се движи, това не влияе на тяхното придвижване и т.н.).

Спецификации на решението

1. Направете проучване в Интернет за това как може да реализирате подобно приложение. Ако използвате WinAPI може би това ще ви бъде от полза. Този блог също би ви бил полезен.

2. Изберете подходящи структури от данни, с които да представите игралното поле и елементите върху него.

3. Изберете подходящи алгоритми, които променят състоянията на игралните елементи.
   - Движението на враговете например може да бъде по най-късия път между тях и играча.
   - Позициите на играча, враговете и падащите торби непрекъснато се променят. Така намереният на дадена стъпка път може да не бъде актуален на по-късен етап.

Както и за проект 3, напишете кратко описание (достатъчно е половин до една страница), в което разкажете как смятате да решите задачата и обосновете решенията си на горните въпроси. Не забравяйте да посочите кои алгоритми и СД ще използвате, а също и опишете накратко архитектурата на приложението, което смятате да реализирате. Качете описанието, например като MS-Word или PDF документ, във формата за предаване на проект и се свържете с екипа. Ще получите от нас потвърждение или забележки за неща, които трябва да се променят. Преминете към реализирането на проекта едва след като бъдат изчистени всички забележки свързани с избрания от вас подход. Ако се затруднявате, прочетете как се правят дизайн документи, например в тази статия.

4. Имплементирайте решението си и подгответе кратко демо на играта за защитата. Опитайте се да имплементирате максимално на брой feature-и от играта.