МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»
(Новосибирский государственный университет, НГУ)
Институт информатики и мехатроники
КАФЕДРА ИНФОРМАТИКИ

ОТЧЕТ КУРСОВОГО ПРОЕКТА

Основы алгоритмизации и программирования

РАЗРАБОТКА ВИДЕОИГРЫ «СОКОБАН»

Преподаватель доцент ИИМ НГУ	Галковская О.Г. «___»_____2022 г.
Студент 1 курса гр. 501 СИК ШВ	Ковальчек А.А. «_14_»___июня___2022 г.

Новосибирск

2022

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

1. ИИМ НГУ – Институт информатики и мехатроники Новосибирского государственного университета;
2. MS – Microsoft;
3. WPF – Windows Presentation Foundation;
4. WF – Windows Forms.

ВВЕДЕНИЕ

Видеоигра – это игра, использующая изображения, сгенерированные электронной аппаратурой. Понятию видеоигры смежно понятие компьютерной игры – программы, служащей для организации игрового процесса. Далее в данной работе слово «видеоигра» употребляется в отношении программ, которые соответствуют как первому определению, так и второму.

Видеоигры как явление существуют уже более полувека. Они могут выполнять различные функции и иметь различные функциональности. Общеизвестна и первична их развлекательная функция. Осуществляя рекреацию многих миллионов людей по всему миру, видеоигры увеличивают производительность их труда. Таким образом, разработка видеоигр жизненно необходима для развития народного хозяйства.

Кроме того, видеоигры могут помогать людям в получении и развитии тех или иных навыков, способностей, знаний. Особенно много в этом смысле видеоигры могут дать детям дошкольного и младшего школьного возраста: игровая деятельность является их основной, при этом игровые механики, которые позволили бы детям освоить необходимые им способности (быстрота реакции, логическое мышление, знания об окружающем их мире), очень просты для разработки.

Процесс создания видеоигр называется разработкой. Разработка игр осуществляется специалистами (геймдизайнерами, программистами, тестировщиками и не только). При разработке видеоигр используются различные инструменты разработчика. Стоит отдельно выделить инструменты, предназначенные специально для разработки видеоигр – шейдеры, графические подсистемы, игровые движки, а также утилиты, разрабатываемые в процессе, такие, как редактор уровней.

Видеоигры делятся на жанры. В частности, существуют игры-головоломки, которые требуют от игрока решать логические задачи. Игры

этого жанра разрабатывали задолго до появления компьютера IBM PC, они всегда были популярны среди широчайших слоёв населения. Особенно эти игры полезны детям, которые с их помощью развивают способность мыслить логически.

Цель проекта – силами подготовленных разработчиков создать игру-головоломку, которая могла бы развлекать потребителя и помогала бы ему освоить логическое мышление. Кроме того, необходимо повысить квалификацию студента Института информатики и мехатроники Новосибирского государственного университета, дав ему задачу разработать игру и таким образом предоставив ему возможность получить опыт разработки, навыки кодирования на языке C Sharp и создания графического интерфейса.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Головоломка – жанр видеоигр, целью которых является решение логических задач, требующих от игрока задействования логического и стратегического мышления, а также интуиции [1].

Предтечей компьютерных головоломок были настольные, графические и механические головоломки. Многие из существующих видеоигр-головоломок являются адаптациями таких игр (пасьянсы, маджонг, кроссворды, кубик Рубика). Однако существуют головоломки, игровые механики которых были разработаны специально для вычислительных машин и могут быть адекватно реализованы только посредством электронных вычислений (тетрис, трёхмерный тетрис, физические головоломки).

Головоломки часто подразделяют на традиционные и физические. Физические головоломки отличаются тем, что используют в своих механиках сложные физические расчёты. Они требуют больше вычислительных ресурсов, чем традиционные головоломки, но любые современные персональные компьютеры способны осуществлять необходимые расчёты достаточно быстро для комфортной игры.

Зачастую игры-головоломки являются казуальными, то есть максимально простыми для освоения и поэтому доступными для широчайшего круга пользователей. Однако существуют головоломки со сложными правилами. Кроме того, механики, характерные для головоломок, иногда вставляют в игры других жанров.

Некоторые игровые механики были реализованы многократно разными разработчиками из разных стран, для разных платформ и в разные времена в виде разных видеоигр-головоломок. Наиболее популярны механики тетриса, пятнашек, двадцати сорока восьми, трёх-в-ряд, классических настольных игр. В некоторых играх-головоломках популярные механики слегка модифицированы.

Игры-головоломки зачастую не требуют хорошей графики. В таком случае их могут реализовать, используя веб-технологии (HTML, PHP, JS) или простейшие инструментарии элементов графического интерфейса (Windows Forms). Однако возможны имплементации с использованием программных интерфейсов трёхмерной графики (OpenGL, DirectX) и игровых движков (Unity).

Стоит также рассмотреть специалиста, которому планируется поручить разработку игры – студента ИИМ НГУ. Этот человек плохо понимает физические процессы, поэтому нежелательно поручать ему разработку физической головоломки. Также он не имеет знаний в области искусственного интеллекта, поэтому парные настольные игры также нежелательны. Кроме того, он считает, что у него нет необходимости изучать движок Unity в настоящий момент. Если поручить ему разработку игры на Unity, его мотивация катастрофически снизится, что может привести к тому, что проект не будет реализован. Наконец, известно, что этому студенту из головоломок наиболее интересны для реализации сокобан и реверси.

2 ПОСТАНОВКА ЗАДАЧИ

Разработать видеоигру-головоломку, воплощающую механики, впервые представленные в игре Хироюки Имабаяши «Сокобан» (японское «Кладовщик») 1982 года выпуска. Допустимо модифицировать механики так, чтобы целевая аудитория проекта (студент ИИМ НГУ и его руководитель) продолжала воспринимать игру как разновидность сокобана.

Механика сокобана такова: ограниченное игровое поле делится на клетки, в клетках могут располагаться объекты: игровой персонаж, ящики, стены, места для расстановки. Игровой персонаж и ящики могут двигаться по клеткам горизонтально и вертикально, не выходя за границы поля и не проходя сквозь стены. При попытке сдвинуть игрового персонажа на клетку, где уже стоит ящик, ящик движется в ту же сторону, если клетка, на которой он должен оказаться, свободна. Победа засчитывается, когда все клетки, на которых есть места для расстановки, также заняты ящиками. Возможна загрузка различных игровых уровней, которые предоставляют разные поля с разной расстановкой объектов. [2]

Рассмотреть уже существующие имплементации, установить, как именно они развлекают потребителя и развивают его логическое мышление. Выделить достоинства и недостатки установленных методов и предложить решения, которые позволили бы разрабатываемому программному средству стать конкурентоспособным.

Использовать модель разработки, которая позволила бы создать план разработки максимально быстро, не возвращаясь к этой задаче впоследствии.

Использовать язык C Sharp для кодирования и какую-либо графическую подсистему для создания графического интерфейса для того, чтобы студент ИИМ НГУ мог получить соответствующие навыки.

3 АНАЛОГИ

Механики сокобана были реализованы много сотен раз. Нецелесообразно рассматривать каждую из них. Достаточно рассмотреть лишь два самых популярных аналога.

1. «Сокобан» Хироюки Имабаяши

Оригинальный сокобан. Разрабатывался для персонального компьютера PC-88. Графика схематичная, спрайтовая, расцветка ограничена восьмибитной архитектурой компьютера. Клетки, на которые нужно ставить ящики, не выделяются и всегда сосредоточены в небольшой комнате.

2. «Ещё один сокобан» Дэвида Джоффа

Свободная кроссплатформенная реализация для Linux, MacOS и Windows. Графика современная, спрайтовая. Правила стандартные. Клетки для ящиков выделены и могут располагаться где угодно. Есть возможность загружать уровни, созданные энтузиастами.

Все приведённые аналоги используют спрайты для того, чтобы отображать объекты, что необязательно для целевой аудитории проекта. Ни один из них не написан на языке C Sharp.

Следует позаимствовать у сокобана Джоффа выделение клеток для ящиков. При этом загрузка уровней не нужна – гораздо проще разработать генератор случайных уровней. Также не нужна спрайтовая графика – целевая аудитория примет графические примитивы, реализовать которые гораздо менее трудозатратно.

Кроме того, правила сокобана следует модифицировать: ящики сдвигаются игровым персонажем даже в том случае, если за ними стоят другие ящики. В таком случае эти ящики тоже сдвигаются, если на их пути нет препятствий. Если же они есть, ни один объект не меняет своего положения.

4 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ

Игрок является единственным пользователем. Он может запустить игру. При запуске генерируется уровень, который отражает прямоугольное игровое поле, состоящее из клеток, каждая из которых имеет декартовы координаты. Затем уровень загружается. После приложение отрисовывает игровое поле с объектами: игровым персонажем, стенами, ящиками и местами для ящиков.

Игрок, нажимая стрелки на клавиатуре, даёт игровому персонажу команду сдвинуться на одну клетку в соответствующем направлении. Если клетка, на которую планируется перемещение, занята стеной или находится за пределами поля, игровой персонаж остаётся на месте. Если там находится ящик, то он перемещается на одну клетку в ту же сторону, что и игрок, но только в том случае, если новая дислокация ящика не занята стеной и не находится за пределами поля. Если на новой дислокации ящика стоит ещё один ящик, процедура проверки доступности клетки повторяется рекурсивно. Объекты сдвигаются лишь в том случае, если планируемое расположение ни одного из них не выпадает на клетку, занятую стеной или находящуюся за пределами игрового поля.

Место для ящика – проходимый объект. Это означает, что клетки, занятые им, проходимы для игрового персонажа и ящиков, и оно при этом не меняет дислокации. Победа в игре засчитывается тогда, когда все места для ящиков, присутствующие на поле, заняты ящиками, то есть нет ни одной клетки с местом для ящика, на котором не было бы ещё и ящика. В этом случае пользователь получает уведомление, и игра заканчивается.

После того, как игрок нажимает какую-либо клавишу, приложение перерисовывает игровое поле, чтобы отразить его текущее состояние. Кроме

того, игрок по ходу игры может дать с клавиатуры команду перегенерировать уровень, чтобы игра началась сначала.

5 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ

5.1 Требования к программному обеспечению

Операционная система семейства Microsoft Windows NT.

5.2 Требования к аппаратному обеспечению

IBM PC-совместимый компьютер, 512 килобайт свободного места на жёстком диске.

5.3 Требования к надёжности

Устойчивость функционирования при наличии ошибок во входных данных, возможность обработки ошибочных ситуаций, наличие тестов для проверки допустимых значений входных данных, наличие средств контроля корректности входных данных.

6 ХАРАКТЕРИСТИКА ВЫБРАННЫХ ПРОГРАММНЫХ СРЕД И СРЕДСТВ

Согласно поставленным задачам, программный продукт должен быть написан на языке C Sharp. Этот язык объектно-ориентирован – таким образом он реализует самую распространённую и удобную на сегодняшний день парадигму программирования. Кроме того, с этим языком связан богатейший инструментарий разработчика, известный как платформа .NET. Библиотеки и подсистемы, входящие в .NET, позволяют легко реализовывать некоторые специфические функциональности, в том числе и графический интерфейс, а компилятор и среды выполнения позволяют сделать программный продукт кроссплатформенным.

Кроме того, должна быть использована графическая подсистема. Была выбрана подсистема Windows Presentation Foundation. В отличие от технологии Windows Forms, WPF использует DirectX с его возможностями аппаратного ускорения графики, из-за чего приложения на WPF выполняются быстрее, чем аналогичные на WF. К сожалению, WPF работает только в системах семейства MS Windows NT, из-за чего преимущество платформы .NET в кроссплатформенности её приложений сходит на нет.

7 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ

Входная точка приложения находится в классе App. Приложение обращается к методу RandomizeLevel класса Randomizer, которое, используя встроенный в .NET класс Random, случайно генерирует ширину и высоту уровня, заполняет его объектами, после чего записывает уровень в файл input в нижеописанном формате (см. часть 8 данной работы). Затем приложение вызывает метод Run класса MainController.

Основной контроллер загружает уровень, обращаясь к методу Load класса LevelLoader. Этот метод читает файл input и инициализирует поля и свойства класса GameController, отвечающие за представление текущего состояния игрового поля, согласно значениям из этого файла. После этого основной контроллер уходит в бесконечный цикл, в котором создаётся и отрисовывается окно класса MainWindow, показывающее игроку текущее состояние игрового поля, после чего методом WaitForTurn класса KeyboardController ожидается нажатие клавиши. Цикл прерывается, когда метод CheckWin класса GameController, отвечающий за проверку текущего состояния на победу, возвращает истину.

Метод WaitForTurn класса KeyboardController запрашивает нажатие клавиши. В случае, если нажимаются стрелки, вызывается метод MovePlayer класса GameController. Если же нажата клавиша R, приложение перегенерирует уровень и перезапускает основной контроллер.

Класс GameController содержит информацию о границах поля, список игровых объектов, а также методы CheckWin, SearchGameObjectByCoordinates и MovePlayer.

Метод SearchGameObjectByCoordinates соотносит данные координаты с объектом, перебирая объекты из списка и сравнивая данные координаты с координатами объектов. Если координаты клетки предполагают, что она находится за пределами игрового поля, метод возвращает объект стены, блокируя таким образом движение.

Метод CheckWin перебирает объекты из списка, после чего, найдя место для ящика, передаёт его координаты поисковому методу. Так как объекты в списке отсортированы загрузчиком уровней так, что ящики поисковым методом находятся всегда вперёд мест для ящиков, можно сделать вывод, что если нашёлся не ящик, то ящика на этом месте нет, следовательно, условие победы не выполнено, и возвращается ложь. Если клеток, где были бы места для ящиков, но не было бы ящиков, не находится, метод возвращает истину.

Метод MovePlayer ищет в списке объектов игрового персонажа и вызывает его метод Move.

Все типы игровых объектов являются наследниками абстрактного класса GameObject. Любой объект имеет декартовы координаты. Объекты типа Goal (места для ящиков) реализуют интерфейс ITransparent, что означает, что их можно пройти насквозь. Объекты типов Player и Box реализуют интерфейс IMovable, и у них есть метод Move.

Метод Move запрашивает поиск объекта, который предположительно находится на клетке, на которую планируется перемещение. Если на этой клетке не пусто, объект не движим и не проходим, исполнение метода завершается с результатом «ложь». Если на этой клетке есть движимый объект (то есть ящик), вызывается его метод Move. Если этот метод возвращает ложь, то и оригинальный метод Move возвращает ложь. Если исполнение метода всё же доходит до конца, координата объекта изменяется и возвращается истина, что говорит о том, что перемещение объекта было успешным.

8 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Уровни генерируются случайно, однако перед загрузкой записываются в файл. Сделано это для того, чтобы впоследствии было проще улучшить игру, добавив в неё функциональность загрузки заранее созданных уровней.

Файл называется input и находится в корневой директории игры. В нём две строки. В первой указаны два целых числа. Первое (xLimit) означает количество клеток, которое игровое поле имеет в ширину, второе (yLimit) – количество клеток в высоту. Вторая строка – набор символов. N-й символ по счёту определяет, какой объект находится в клетке с координатами ((N — 1) % xLimit; (N — 1) / xLimit): игровой персонаж („p“), стена („w“), ящик („b“), место для ящика („g“) или никакого („e“). Возможность поставить на клетку сразу несколько объектов при загрузке уровня не предусмотрена.

На выходе игра даёт окна, каждое из которых отражает текущее состояние игрового поля. Каждая клетка представлена в виде монохромного квадрата. Цвет квадрата зависит от того, какой объект находится на клетке: зелёный, если игровой персонаж, багряный, если ящик, красный, если стена, бирюзовый, если место для ящика, и серый, если никакого. Клетки за пределами игрового поля окрашены в белый цвет.

9 ТЕСТИРОВАНИЕ И ОТЛАДКА

Были разработаны тесты для того, чтобы проверить, насколько игровое приложение соответствует функциональным и нефункциональным требованиям, при необходимости доработать его и выпустить в надлежащем качестве.

Тест №1: запуск приложения.

Входные данные: пользователь запускает исполняемый файл SokobanGraphical.exe.

Ожидаемый результат: отображается консоль и окно с загруженным уровнем.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №2: движение игрового персонажа.

Входные данные: игрок нажимает на клавиши со стрелками.

Ожидаемый результат: игровой персонаж перемещается по полю.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №3: сдвиг ящика на свободную клетку.

Входные данные: игрок сдвигает игрового персонажа на клетку с ящиком, при этом следующая клетка свободна.

Ожидаемый результат: игровой персонаж и ящик сдвигаются на одну клетку.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №4: сдвиг игрового персонажа на стену.

Входные данные: игрок пытается сдвинуть игрового персонажа на клетку со стеной.

Ожидаемый результат: игровой персонаж остаётся на месте.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №5: сдвиг игрового персонажа на клетку за пределами игрового поля.

Входные данные: игрок сдвигает игрового персонажа на клетку за пределами игрового поля.

Ожидаемый результат: игровой персонаж и ящик сдвигаются на одну клетку.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №6: сдвиг ящика на непроходимую клетку.

Входные данные: игрок сдвигает игрового персонажа на клетку с ящиком, при этом следующая клетка непроходима

Ожидаемый результат: игровой персонаж и ящик остаются на месте.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №7: успешный сдвиг цепочки ящиков.

Входные данные: игрок сдвигает игрового персонажа на клетку с ящиком, при этом следующие клетка заняты ящиками. Клетка, следующая за последним ящиком, свободна.

Ожидаемый результат: игровой персонаж и все ящики сдвигаются.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

Тест №8: безуспешный сдвиг цепочки ящиков.

Входные данные: игрок сдвигает игрового персонажа на клетку с ящиком, при этом следующие клетка заняты ящиками. Клетка, следующая за последним ящиком, непроходима.

Ожидаемый результат: игровой персонаж и все ящики остаются на месте.

Полученный результат: совпадает с ожидаемым.

Тест пройден успешно.

10 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Начните игру, запустив исполняемый файл SokobanGraphical.exe.

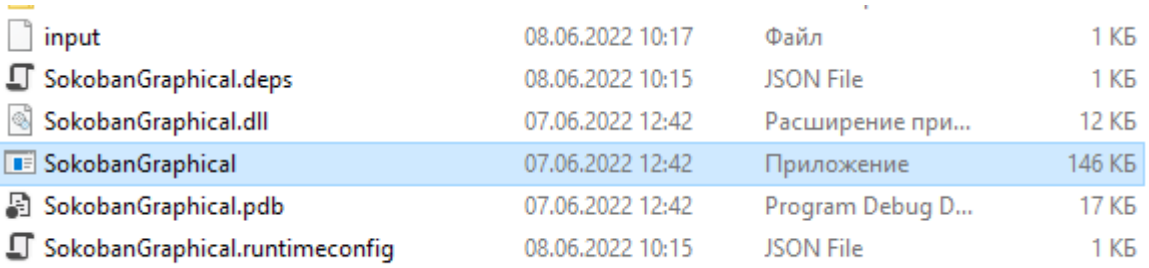

Рисунок 1 – Запуск приложения «Сокобан»

Удостоверьтесь, что окно консоли активно. Нажимайте стрелки на клавиатуре, чтобы сдвинуть игрока.

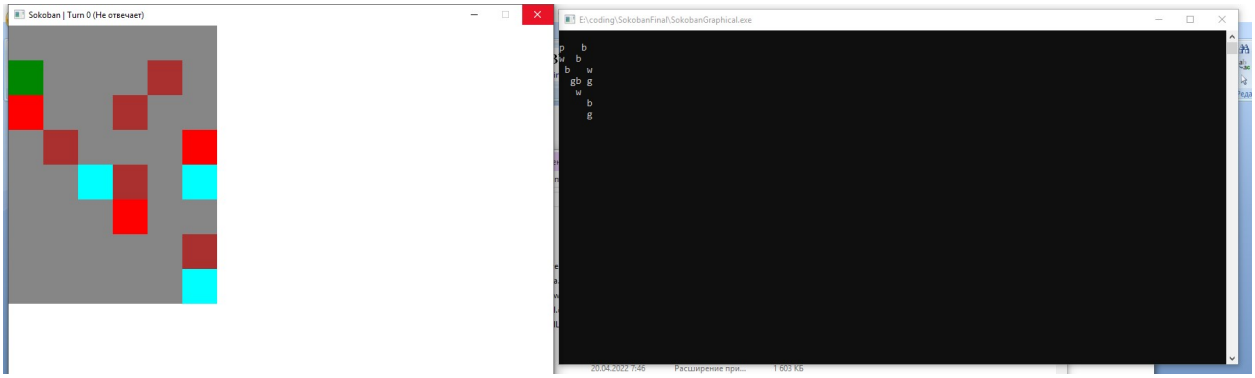

Рисунок 2 – Графический и консольный интерфейс приложения «Сокобан»

Если уровень не устраивает вас, вы можете перегенерировать его, нажав кнопку R на клавиатуре.

Делайте ходы до тех пор, пока все места для ящиков не окажутся заняты ящиками. Вы увидите уведомление о победе. После нажатия кнопки «OK» приложение закроет все окна и завершит работу.

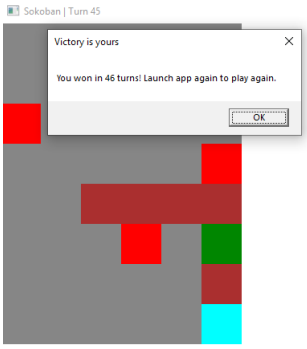

Рисунок 3 – Уведомление о победе

ЗАКЛЮЧЕНИЕ

Была успешно разработана видеоигра «Сокобан». Развлекательная функция данного программного продукта не уступает соответствующей функции аналогов благодаря случайному генерированию уровней и модификации правил. То же можно сказать о развивающей функции.

Студент ИИМ НГУ получил опыт разработки, навыки кодирования на языке C Sharp и создания графического интерфейса. Тем самым он повысил свою квалификацию, увеличив шансы улучшить свой уровень жизни в ближайшие годы и внести свой вклад в развитие народного хозяйства.

Было обеспечено разбиение на модули, что улучшит сопровождаемость в случае, если в будущем будет принято решение улучшить эту видеоигру, портировать её на другие платформы. Кроме того, реализованные запись уровня в файл и чтение уровня из файла позволят легко разработать редактор уровней и загрузчик заранее заготовленных уровней.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Википедия, свободная энциклопедия [Электронный ресурс] — URL : https://ru.wikipedia.org/wiki/Головоломка_(жанр_компьютерных_игр) — (Дата обращения: 14.06.2022);

2 Википедия, свободная энциклопедия [Электронный ресурс] — URL : https://ru.wikipedia.org/wiki/Сокобан — (Дата обращения: 14.06.2022);

ПРИЛОЖЕНИЕ А

```
public partial class App : Application
   {
      void AppStartup(object sender, StartupEventArgs e)
      {
         string levelPath = "input";
         Randomizer.RandomizeLevel(levelPath);
         MainController.Run(levelPath);
      }
   }


static class ConsoleDrawer
{
   public static void Draw()
   {
      Console.Clear();
      for (int i = 0; i < GameController.yLimit; i++)
      {
         for (int j = 0; j < GameController.xLimit; j++)
         {
             GameObject obj = GameController.SearchGameObjectByCoordinates(j,
i);
            if (obj is Player) Console.Write('p');
            if (obj is Box) Console.Write('b');
            if (obj is Goal) Console.Write('g');
            if (obj is Wall) Console.Write('w');
            if (obj is null) Console.Write(' ');
         }
         Console.WriteLine();
```

```
        }
    }
}

static class GameController
{
    public static GameObject[] listOfObjects;
    public static int xLimit;
    public static int yLimit;
    public static void MovePlayer(Directions dir)
    {
        Player player = listOfObjects[listOfObjects.Length - 1] as Player;
        player.Move(dir);
    }
    public static bool CheckWin()
    {
        foreach (var item in listOfObjects)
        {
            if (item is Goal)
            {
                    if (SearchGameObjectByCoordinates(item.x, item.y) is Goal) return
false;
            }
        }
        return true;
    }
    public static GameObject SearchGameObjectByCoordinates(int x, int y)
    {
        if (x < 0 || y < 0 || x >= xLimit || y >= yLimit) return new Wall();
        foreach (var item in listOfObjects)
```

```
        {
            if (item.x == x && item.y == y)
            {
                return item;
            }
        }
        return null;
    }
}

static class KeyboardController
{
    public static void WaitForTurn()
    {
            switch (Console.ReadKey().Key)
            {
                case ConsoleKey.LeftArrow:
                    GameController.MovePlayer(Directions.Left);
                    break;

                case ConsoleKey.UpArrow:
                    GameController.MovePlayer(Directions.Up);
                    break;

                case ConsoleKey.RightArrow:
                    GameController.MovePlayer(Directions.Right);
                    break;

                case ConsoleKey.DownArrow:
                    GameController.MovePlayer(Directions.Down);
```

```
                break;

            case ConsoleKey.R:
                string levelPath = "input";
                Randomizer.RandomizeLevel(levelPath);
                MainController.Run(levelPath);
                break;

            default:
                break;
        }
    }
}

static class LevelLoader
{
    public static void Load(string path)
    {
        using (StreamReader reader = new StreamReader(path))
        {
            string[] size = reader.ReadLine().Split(' ', 2);
            int xLimit = Convert.ToInt32(size[0]);
            int yLimit = Convert.ToInt32(size[1]);

            GameController.xLimit = xLimit;
            GameController.yLimit = yLimit;

            List<Box> boxes = new List<Box>();
            List<Wall> walls = new List<Wall>();
            List<Goal> goals = new List<Goal>();
```

```
Player player = null;

for (int i = 0; i < yLimit; i++)
{
   for (int j = 0; j < xLimit; j++)
   {
      char input = (char)reader.Read();
      switch (input)
      {
         case 'p':
            player = new Player(j, i);
            break;
         case 'b':
            boxes.Add(new Box(j, i));
            break;
         case 'w':
            walls.Add(new Wall(j, i));
            break;
         case 'g':
            goals.Add(new Goal(j, i));
            break;
         default:
            break;
      }
   }
}

List<GameObject> list = new List<GameObject>();
list.AddRange(boxes);
list.AddRange(goals);
```

```
            list.AddRange(walls);
            list.Add(player);
            GameController.listOfObjects = list.ToArray();
        }
    }
}

static class MainController
{
    public static void Run(string input)
    {
        LevelLoader.Load(input);
        int i = 0;
        while (true)
        {
            ConsoleDrawer.Draw();
            MainWindow wnd = new MainWindow();
                        wnd.Title = $"Sokoban | Turn {i}";
            wnd.Top = 0;
            wnd.Left = 0;
                        wnd.Show();
            KeyboardController.WaitForTurn();
            //wnd.Close();
            i++;
            if(GameController.CheckWin()) break;
        }
            MessageBox.Show($"You won in {i} turns! Launch app again to play again.", "Victory is yours");
        System.Environment.Exit(0);
    }
```

```
}

public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         Run();
      }

      public void Run()
      {
         Draw();
      }

      public void Draw()
      {
         Canvas canvas = new Canvas();

         for (int i = 0; i < GameController.yLimit; i++)
         {
            for (int j = 0; j < GameController.xLimit; j++)
            {
               Rectangle r = new Rectangle();
               r.Width = 50;
               r.Height = 50;
               r.VerticalAlignment = VerticalAlignment.Top;
               Canvas.SetTop(r, i * 50);
               Canvas.SetLeft(r, j * 50);
```

```
                                                    GameObject obj =
GameController.SearchGameObjectByCoordinates(j, i);
                if (obj is Player)
                {
                    r.Fill = Brushes.Green;
                }
                if (obj is Box)
                {
                    r.Fill = Brushes.Brown;
                }
                if (obj is Goal)
                {
                    r.Fill = Brushes.Cyan;
                }
                if (obj is Wall)
                {
                    r.Fill = Brushes.Red;
                }
                if (obj is null)
                {
                    r.Fill = Brushes.Gray;
                }
                canvas.Children.Add(r);
            }
        }
        this.Content = canvas;
    }
}

static class Randomizer
```

```
{
    public static void RandomizeLevel(string path)
    {
        using (StreamWriter writer = new StreamWriter(path, false))
        {
            Random rnd = new Random();
            int xLimit = rnd.Next(5, 10);
            int yLimit = rnd.Next(5, 10);
            writer.Write(xLimit + " " + yLimit);
            writer.WriteLine();
            bool playerHere = false;
            for (int i = 0; i < xLimit * yLimit; i++)
            {
                switch (rnd.Next(15))
                {
                    case 0:
                        if (!playerHere) writer.Write('p');
                        else writer.Write('e');
                        playerHere = true;
                        break;
                    case 1:
                        writer.Write('b');
                        break;
                    case 2:
                        writer.Write('g');
                        break;
                    case 3:
                        writer.Write('w');
                        break;
                    case int n when (n > 3):
```

```
                    writer.Write('e');
                    break;
                default:
                    break;
            }
        }
        if (!playerHere) RandomizeLevel(path);
    }
  }

class Box: GameObject, IMovable
{
  public Box(int x, int y): base(x, y)
  {

  }

  public bool Move(Directions dir)
  {
    GameObject obj;
    switch (dir)
    {
      case Directions.Left:
        obj = GameController.SearchGameObjectByCoordinates(x - 1, y); //what if box is on goal? boxes must stay before in list. and only singleplayer.
            if (obj != null && obj is not ITransparent && obj is not IMovable) return false;
        if (obj is IMovable)
        {
```

```
            IMovable movableObj = obj as IMovable;
            if(!movableObj.Move(Directions.Left)) return false;
        }
        x--;
        break;

    case Directions.Up:
        obj = GameController.SearchGameObjectByCoordinates(x, y - 1); //what if box is on goal? boxes must stay before in list. and only singleplayer.
            if (obj != null && obj is not ITransparent && obj is not IMovable) return false;
        if (obj is IMovable)
        {
            IMovable movableObj = obj as IMovable;
            if(!movableObj.Move(Directions.Up)) return false;
        }
        y--;
        break;

    case Directions.Right:
            obj = GameController.SearchGameObjectByCoordinates(x + 1, y); //what if box is on goal? boxes must stay before in list. and only singleplayer.
            if (obj != null && obj is not ITransparent && obj is not IMovable) return false;
        if (obj is IMovable)
        {
            IMovable movableObj = obj as IMovable;
            if(!movableObj.Move(Directions.Right)) return false;
        }
        x++;
```

```
            break;

        case Directions.Down:
                obj = GameController.SearchGameObjectByCoordinates(x, y + 1);
//what if box is on goal? boxes must stay before in list. and only singleplayer.
                if (obj != null && obj is not ITransparent && obj is not IMovable)
return false;
            if (obj is IMovable)
            {
               IMovable movableObj = obj as IMovable;
               if(!movableObj.Move(Directions.Down)) return false;
            }
            y++;
            break;

        default:
            break;
      }
      return true;
   }
}

enum Directions
{
   Left, Up, Right, Down
}

abstract class GameObject
{
   public int x;
```

```
    public int y;

    public GameObject(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public GameObject()
    {

    }
}

class Goal: GameObject, ITransparent
{
    public Goal(int x, int y): base(x, y)
    {

    }
}

interface IMovable
{
    bool Move(Directions dir);
}

interface ITransparent
{
}
```

```
class Player: GameObject, IMovable
{
   public Player(int x, int y): base(x, y)
   {

   }

   public bool Move(Directions dir)
   {
      GameObject obj;
      switch (dir)
      {
         case Directions.Left:
            obj = GameController.SearchGameObjectByCoordinates(x - 1, y); //what if box is on goal? boxes must stay before in list. and only singleplayer.
               if (obj != null && obj is not ITransparent && obj is not IMovable) return false;
            if (obj is IMovable)
            {
               IMovable movableObj = obj as IMovable;
               if(!movableObj.Move(Directions.Left)) return false;
            }
            x--;
            break;

         case Directions.Up:
            obj = GameController.SearchGameObjectByCoordinates(x, y - 1); //what if box is on goal? boxes must stay before in list. and only singleplayer.
               if (obj != null && obj is not ITransparent && obj is not IMovable) return false;
```

```
            if (obj is IMovable)
            {
               IMovable movableObj = obj as IMovable;
               if(!movableObj.Move(Directions.Up)) return false;
            }
            y--;
            break;

        case Directions.Right:
                obj = GameController.SearchGameObjectByCoordinates(x + 1, y);
//what if box is on goal? boxes must stay before in list. and only singleplayer.
                if (obj != null && obj is not ITransparent && obj is not IMovable)
return false;
            if (obj is IMovable)
            {
               IMovable movableObj = obj as IMovable;
               if(!movableObj.Move(Directions.Right)) return false;
            }
            x++;
            break;

        case Directions.Down:
                obj = GameController.SearchGameObjectByCoordinates(x, y + 1);
//what if box is on goal? boxes must stay before in list. and only singleplayer.
                if (obj != null && obj is not ITransparent && obj is not IMovable)
return false;
            if (obj is IMovable)
            {
               IMovable movableObj = obj as IMovable;
               if(!movableObj.Move(Directions.Down)) return false;
```

```
            }
            y++;
            break;

        default:
            break;
      }
      return true;
   }
}

class Wall: GameObject
{
   public Wall()
   {

   }
   public Wall(int x, int y): base(x, y)
   {

   }
}
```