МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Работа допущена к защите

_____Руководитель

«____»_____20___г.

КУРСОВОЙ ПРОЕКТ

по дисциплине «Выпуск и сопровождение программных продуктов»

на тему: «Разработка системы контроля версий состояний для игры «Выход из комнаты»»

Студент Шорин В.Д.

Шифр 171406

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 71-ПГ

Руководитель _____ Лукьянов П.В.

Оценка: «_____» Дата ___

Орел 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

УТВЕРЖДАЮ :
_____Зав. кафедрой
«___»_____20__г.

ЗАДАНИЕ
на курсовой проект

по дисциплине «Выпуск и сопровождение программных продуктов»

Студент Шорин В.Д. Шифр 171406
Институт приборостроения, автоматизации и информационных технологий
Направление подготовки 09.03.04 «Программная инженерия»
Группа 71ПГ

1.Тема курсовой работы
«Разработка системы контроля версий состояний для игры «Выход из комнаты»

2.Срок сдачи студентом законченной работы «15» декабря 2020

3.Исходные данные

Описать предметную область для системы контроля версий состояний игры «Выход из комнаты»

Разработать модель системы контроля версий

Спроектировать алгоритмы, основанные на разработанной модели

Реализовать систему контроля версий состояний игры «Выход из комнаты»

4.Содержание курсовой работы

Описание предметной области для системы контроля версий состояний игры «Выход из комнаты»

Разработка модели системы контроля версий

Проектирование алгоритмов, основанных на разработанной модели

Реализация системы контроля версий состояний игры «Выход из комнаты»

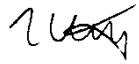
Описание пользовательского интерфейса

5.Отчетный материал курсовой работы

Пояснительная записка курсовой работы; приложение, записанное на CD-диске

Руководитель _____Лукьянов П.В.

Задание принял к исполнению: «1» октября 2019

Подпись студента 

Содержание

ВВЕДЕНИЕ

Система управления (контроля) версиями – программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Каждый пользователь в процессе создания каких-либо моделей, документов или в процессе развлечений (например, игр) совершает какие-либо ошибки, неправильные последовательности действий и хочет вернуть назад проделанные изменения.

Актуальность данной тематики выражена в том, что система контроля версий хранит различные состояния соответствующей программы (модели, документа, состояния игры) и позволяет пользователю в любой момент вернуться к определенному состоянию (сохранению в игре).

Целью данной курсовой работы является разработка системы контроля версий состояний для игры «Выход из комнаты».

Задачами данной курсовой работы, которые необходимо выполнить для достижения поставленной цели, являются:

- Описание предметной области для системы контроля версий состояний игры «Выход из комнаты»;
- Разработка модели системы контроля версий;
- Проектирование алгоритмов, основанных на разработанной модели;
- Реализация алгоритмов системы контроля версий состояний игры «Выход из комнаты»;
- Описание пользовательского интерфейса.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ДЛЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ СОСТОЯНИЙ

В качестве программы, для которой будет разрабатываться система контроля версий состояний в данной курсовой работе рассматривается игра «Выход из комнаты». «Выход из комнаты» - логическая игра-головоломка, в которой в комнате, показанной в виде плана, находятся игрок, ящики, плитки с крестами, являющимися конечными позициями для ящиков и дверь. Цель игрока – поставить ящики, передвигая их по комнате, до их конечных позиций (плитки с крестами), после чего открывается дверь и игрок может перейти на следующий уровень.

Соответственно, в процессе прохождения уровня пользователь может либо случайно передвинуть ящик не в ту позицию, в которую хотел изначально, либо совершить последовательность действий, которая не привела его к выполнению поставленной цели, либо любое другое нежелательное действие.

Если в данную игру внедрить систему контроля версий состояний, то пользователю откроется возможность сохранения позиций ящиков и игрока вплоть до каждого шага, что позволит ему отыскать выигрышную стратегию без необходимости перезагрузки уровня после каждой ошибки. А также это откроет возможность более безболезненного поиска альтернативных стратегий достижения цели.

Для выполнения поставленной задачи и реализации ПО и проектируемых алгоритмов будет использоваться среда Unity, являющаяся межплатформенная средой разработки компьютерных игр, так как она предоставляет множество функциональных возможностей для конечного программного продукта [1].

В качестве языка программирования будет использован язык C# – один из двух официально существующих вариантов выбора для разработки в среде Unity и одновременно с этим наиболее широко используемый и поддерживаемый язык в указанной среде [3].

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ СОСТОЯНИЙ

Система контроля версий состояний будет состоять из модели и модуля взаимодействия с ней.

2.1 Разработка модели

В качестве модели для системы контроля версий состояний для игры «Выход из комнаты» выберем структуру данных «дерево».

Дерево – одна из наиболее широко распространённых структур данных в информатике, эмулирующая древовидную структуру в виде набора связанных узлов. Является связным графом, не содержащим циклы.

Вместе с термином «дерево» применяются следующие понятия:

- Корневой узел – в самый верхний узел дерева;
- Корень – одна из вершин, по желанию наблюдателя;
- Лист, листовой или терминальный узел – узел, не имеющий дочерних элементов;
- Внутренний узел – любой узел дерева, имеющий потомков, и таким образом, не являющийся листовым узлом.

Каждый узел имеет неограниченное количество потомков. При удалении какого-либо узла все его потомки становятся потомками родителя удаляемого узла [2].

Таким образом, исходя из полученной информации о структуре данных типа «дерева», мы можем убедиться, что она напрямую связано с поставленной задачей создания системы контроля версий состояний. Убедиться в данном выводе мы можем, основываясь на том факте, что при сохранении состояний нам важно иметь возможность сохранения неограниченного количества состояний. Также, в каждом состоянии мы должны иметь возможность сохранения такого же неограниченного количества состояний.

В реализуемой программной системе (ПС) корневым узлом будет являться сохранение данных при первом запуске игровой сцены. Внутренними узлами, соответственно, будут являться все последующие сохранения, а листами сохранения без потомков.

Создадим и опишем соответствующую создаваемой модели диаграмму классов (рисунок 2.1).

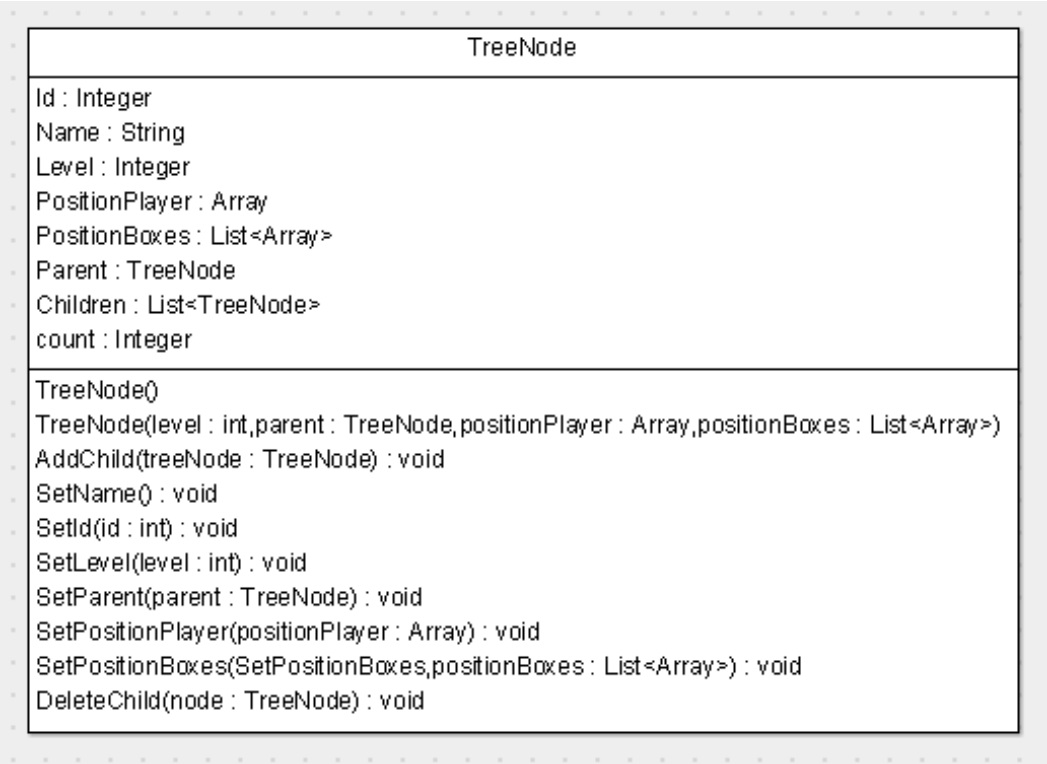


Рисунок 2.1 – Диаграмма классов разрабатываемой модели

Класс TreeNode представляет собой узел дерева. Имеет следующие атрибуты:

- Id: Integer (get; private set;) – порядковый номер сохранения. Доступно для чтения, но скрыто для модификации извне;
- Name: String (get; private set;) – строка с названием сохранения (формируется как строка «Save_» с добавлением времени создания сохранения и полученным номером. Доступно для чтения, но скрыто для модификации извне;
- Level: Integer (get; private set;) – номер уровня, на котором было сделано сохранение. Доступно для чтения, но скрыто для модификации извне;

• PositionPlayer: Array (get; private set;) – массив координат (x, y) позиции игрока в момент сохранения. Доступно для чтения, но скрыто для модификации извне;

• PositionBoxes: List<Array> (get; private set;) – список координат позиций ящик на уровне в момент сохранения. Доступно для чтения, но скрыто для модификации извне;

• Parent: TreeNode (get; private set;) – ссылка на родителя (предыдущее сохранение, после которого было сделано текущее). Доступно для чтения, но скрыто для модификации извне;

• Children: List<TreeNode> (get; private set;) – список ссылок на детей (последующие сохранения, сделанные после текущего). Доступно для чтения, но скрыто для модификации извне;

• Count: Integer (private static)– статичное скрытое поле, которое автоматически увеличивается при создании нового экземпляра класс. По сути, автоматический счетчик порядкового номера (поле Id) Доступно для чтения и изменения только внутри класса.

Данный класс обладает следующими методами:

• TreeNode (): public – пустой конструктор класса;

• TreeNode (level: integer, parent: TreeNode, positionPlayer: Array, positionBoxes: List<Array>): public – конструктор класса, принимающий параметры сохранения (номер уровня, ссылку на родителя, координаты позиции игрока, список координат позиций ящиков) и вызывающий соответствующие методы, устанавливающие эти значения в соответствующие поля;

• AddChild (treeNode: TreeNode): public void – добавляет потомка (следующее сохранение) текущему сохранению;

• SetName (): public void – устанавливает поле Name;

• SetId (id: integer): public void – устанавливает поле Id в соответствии с полученным значением;

• SetLevel (level: integer): public void – устанавливает поле Level в соответствии с полученным значением;

• SetParent (parent: TreeNode): public void – устанавливает поле Parent в соответствии с полученным значением;

• SetPositionPlayer (positionPlayer: Array): public void – устанавливает поле PositionPlayer в соответствии с полученным значением;

• SetPositionBoxes (positionBoxes: List<Array>): public void – устанавливает поле PositionBoxes в соответствии с полученным значением;

• DeleteChild (node: TreeNode): public void – удаляет из списка потомком полученное значение.

2.2 Проектирование модуля взаимодействия с моделью

Для работы с созданным классом TreeNode создадим и опишем необходимые классы (рисунок 2.1).

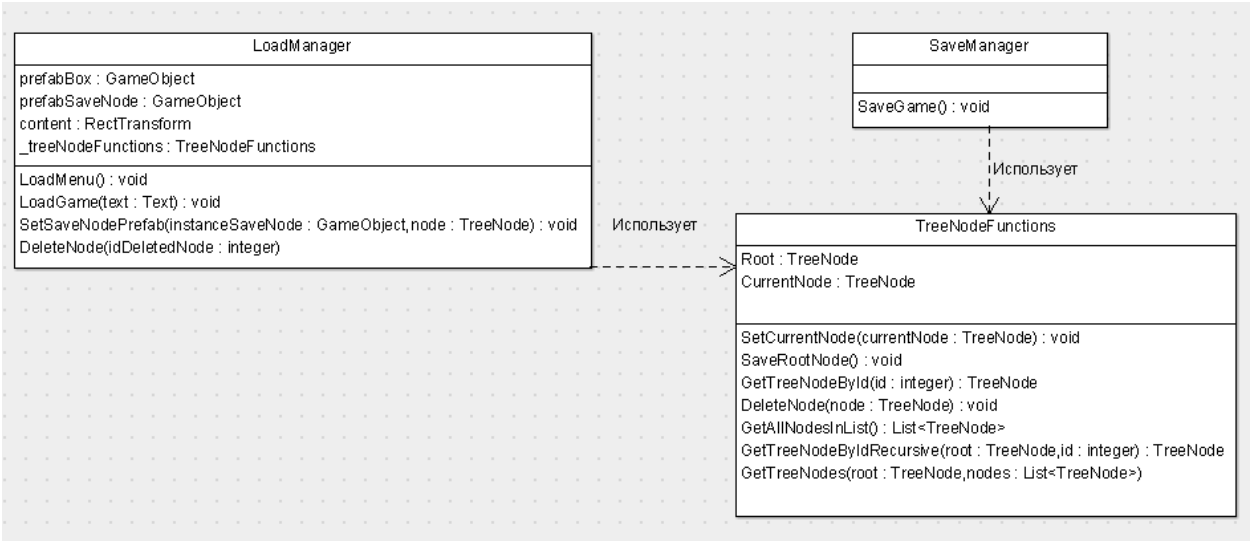


Рисунок 2.1 – Диаграмма классов модуля взаимодействия с моделью

На диаграмме представлены три класса: LoadManager, SaveManager и TreeNodeFunctions. Опишем их по порядку.

Класс LoadManager имеет следующие поля:

• prefabBox: Gameobject – ссылка на экземпляр ящика. Скрытое поле, доступное для обращения и изменения только внутри класса;

• prefabSaveNode: Gameobject – ссылка на экземпляр компонента пользовательского интерфейса, отвечающего за вывод компонента (рисунок 2.2). Скрытое поле, доступное для обращения и изменения только внутри класса

• content: RectTransform – ссылка на компонент пользовательского интерфейса в котором будут отображаться все компоненты с сохранениями. Скрытое поле, доступное для обращения и изменения только внутри класса;

• _treeNodeFunctions: TreeNodeFunctions – ссылка на класс, содержащий функции по работе с моделью. Скрытое поле, доступное для обращения и изменения только внутри класса.

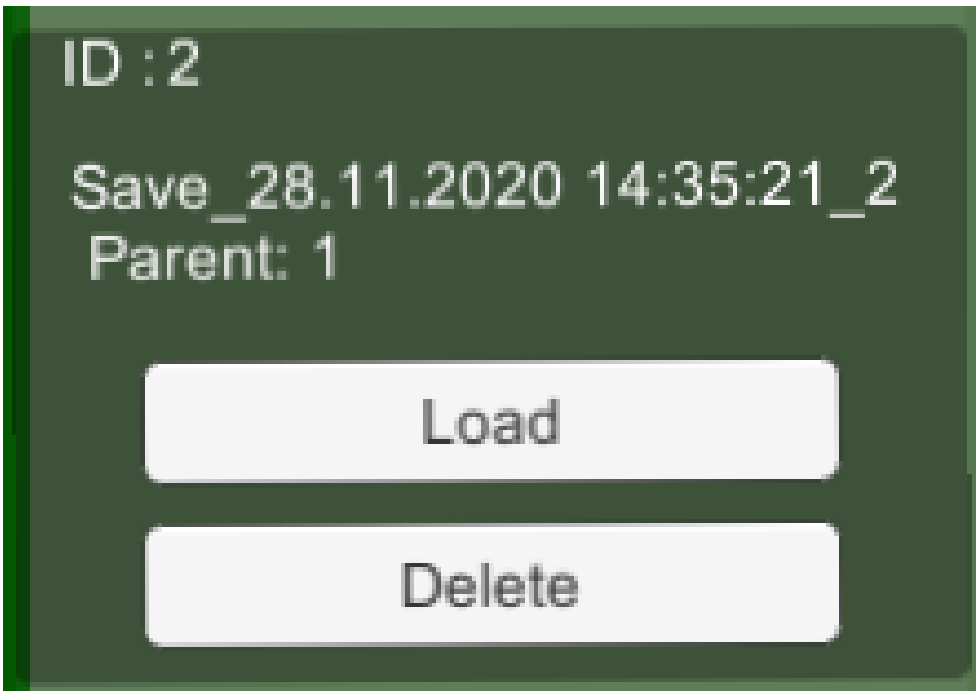


Рисунок 2.2 – Экземпляр компонента вывода одного сохранения

Методы, представленные в классе LoadManager:

• LoadMenu (): public void – загружает список всех сохранений в компонент пользовательского интерфейса content;

• LoadGame (text: Text): public void – загружает состояние из соответствующего сохранения;

• SetSaveNodePrefab (): private void – устанавливает в шаблон компонента сохраненного состояния соответствующие значения;

• DeleteNode (idDeletedNode: Integer): private void – удаляет сохраненное состояние по соответствующему порядковому номеру.

Опишем класс SaveManager. Данный класс представлен одним методом:

• SaveGame (): public void – сохраняет текущее состояние игры в модель.

Класс TreeNodeFunctions представлен следующими полями и методами:

• Root: public static TreeNode – хранит корневой узел модели, соответствующий первому сохраненному состоянию игры при загрузке сцены;

• CurrentNode: public static TreeNode – хранит ссылку на элемент модели, соответствующий текущему состоянию игры (последнее сохранение);

• SetCurrentNode (currentNode: TreeNode) : public static void – устанавливает значение текущего состояния игры (сохранения) в переданное;

• SaveRootNode (): public static void – сохраняет начальное состояние игры в корневой узел;

• GetTreeNodeById (id: integer): public static TreeNode – вызов рекурсивного метода GetTreeNodeByIdRecursive для возврата состояния игры по порядковому номеру узла;

• DeleteNode (node: TreeNode): public static void – удаляет переданное значение состояния из дерева;

• GetAllNodesInList (): public List<TreeNode> – возвращает дерево состояний как список;

• GetTreeNodeByIdRecursive (root: TreeNode, id: integer): private static TreeNode – рекурсивно ищет узел в дереве по переданному порядковому номеру;

• GetTreeNodes (root: TreeNode, nodes: ref List<TreeNode>): private void – добавляет в передаваемый по ссылке список nodes все записи состояний в дереве.

3 ПРОЕКТИРОВАНИЕ АЛГОРИТМОВ ДЛЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ СОСТОЯНИЙ

Опишем алгоритмы, необходимые для работы с разработанной моделью.

Удаление записи из дерева (DeleteNode). На вход алгоритма поступает запись, которую необходимо удалить. Сначала из этой записи получаем ее родителя и устанавливаем, как текущую. Затем идем в цикле по потомкам переданной записи, добавляем каждую запись как потомка родителя переданной записи и, также, назначаем каждой записи родителя. В конце алгоритма удаляем у родителя переданную запись. Схема данного алгоритма представлена на рисунке 3.1.

Рисунок 3.1 – Схема алгоритма удаления записи

Получение записи по ее порядковому номеру (Рекурсивная функция GetTreeNodeByIdRecursive). Проверить номер полученного корня на соответствие искомому. Если они равны, то вернуть корень, иначе идем по потомкам и для каждого вызываем эту же функцию рекурсивно и проверяем результат этого вызова на пустоту. Если пришел непустой результат, то возвращаем его. В конце функции возвращаем пустоту. Схема данного алгоритма представлена на рисунке 3.2.

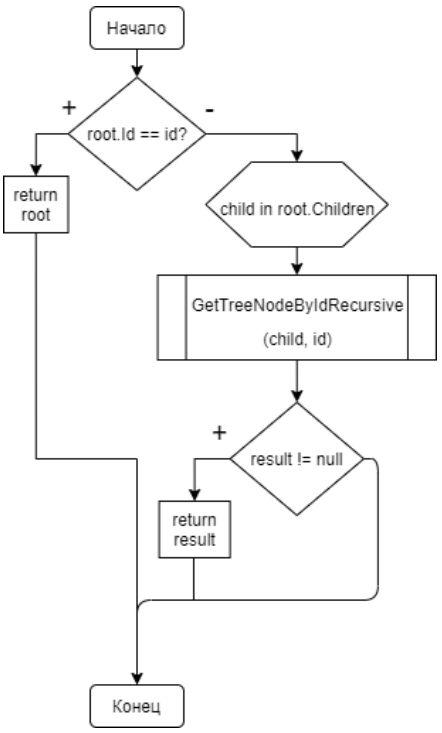


Рисунок 3.2 – Схема алгоритма поиска записи по номеру

Получение дерева как списка его узлов (Рекурсивная функция GetTreeNodes). Входными параметрами являются ссылка на корень дерева (поддерева) и передаваемый по ссылке список записей, в который добавляются новые записи из дерева. Сначала проверяем корень на пустоту: если не пустой, то заносим его в список. Далее идем по списку потомков переданного корня и вызываем для них текущую функцию рекурсивно. Схема данного алгоритма представлена на рисунке 3.3.

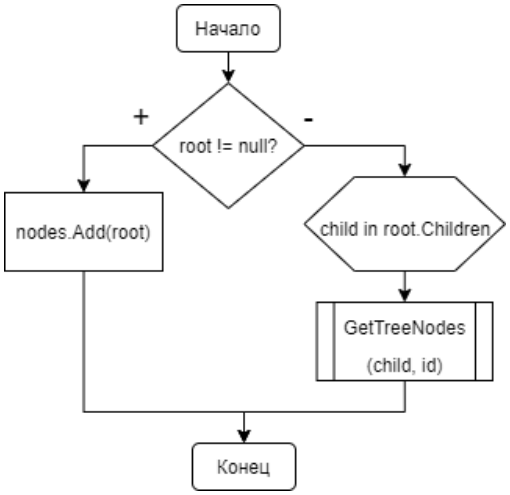


Рисунок 3.3 Схема алгоритма составления списка записей из дерева

Сохранение состояния игры (SaveGame). Сначала получаем позиции игрока и ящиков, запоминаем уровень, на котором находится игрок. Затем создаем экземпляр новой записи для дерева, добавляем его как потомка текущей записи (состояния игры). В конце устанавливаем созданный экземпляр как текущий. Схема данного алгоритма представлена на рисунке 3.4.

Загрузка игры (LoadGame). Сначала получаем порядковый номер загружаемого состояния и соответствующую ему запись в модели. Устанавливаем значение загружаемого уровня. Затем находим игрока и устанавливаем его позицию в загруженную. Находим все старые ящики и заменяем их на новые. В конце устанавливаем загруженное состояние игры в текущее. Схема данного алгоритма представлена на рисунке 3.5.

Установка значений компонента сохраненного состояния в пользовательском интерфейсе (SetSaveNodePrefab). На вход подается шаблон компонента и устанавливаемая запись. Сначала получаем и устанавливаем порядковый номер записи. Затем делаем тоже самое с порядковым номером родителя. Также, устанавливаем название записи. Добавляем функцию LoadGame как слушателя события нажатия по кнопке загрузки «Load» (рисунок 2.2). Проверяем, является ли устанавливаемая запись корневым узлом. Если да, то убираем кнопку удаления записи «Delete» (рисунок 2.2), иначе добавляем функцию DeleteNode как слушателя события нажатия на кнопку удаления записи. Схема данного алгоритма представлена на рисунке 3.6.

Рисунок 3.4 – Схема алгоритма сохранения состояния игры

Рисунок 3.5 Схем алгоритма загрузки состояния игры

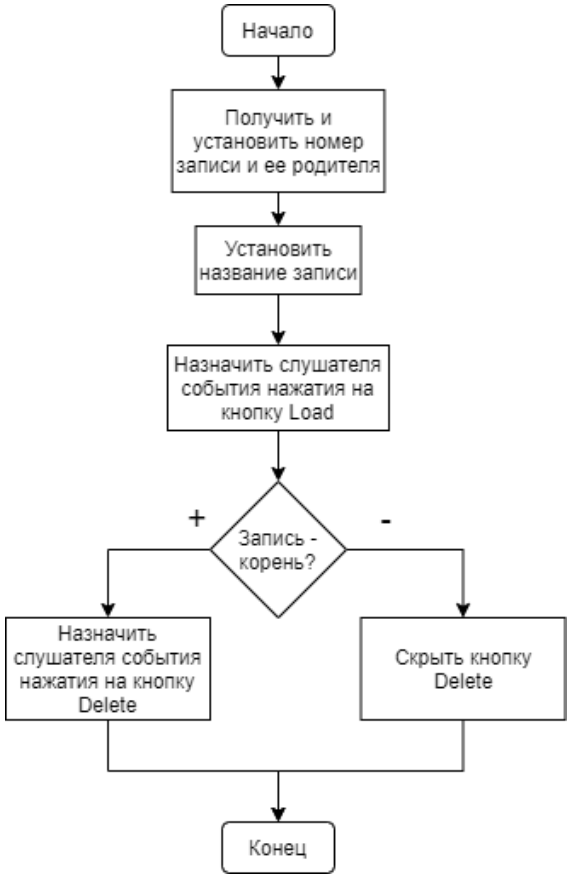


Рисунок 3.6 – Схема алгоритма установки значений шаблона компонента записи

4 РЕАЛИЗАЦИЯ АЛГОРИТМОВ ДЛЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ СОСТОЯНИЙ

Описанные в предыдущей главе алгоритмы реализованы в соответствующих по названию функциях в программной системе.

Код реализации алгоритма удаления записи из дерева представлен ниже:

```
private void DeleteNode (int idDeletedNode) {
        TreeNode deleteNode = TreeNodeFunctions.GetTreeNodeById
(idDeletedNode);
        TreeNodeFunctions.DeleteNode (deleteNode);
        LoadMenu ();
    }
```

Код реализации алгоритма получения записи по ее порядковому номеру представлен ниже:

```
private static TreeNode GetTreeNodeByIdRecursive(TreeNode root, int id) {
    if (root.Id == id) {
        return root;
    } else {
        foreach (var child in root.Children) {
            TreeNode result = GetTreeNodeByIdRecursive(child, id);
            if (result != null) {
                return result;
            }
        }
    }
    return null;
}
```

Код реализации алгоритма получения дерева как списка его узлов представлен ниже:

```
private void GetTreeNodes(TreeNode root, ref List<TreeNode> nodes) {
        if (root != null) {
            nodes.Add(root);
        }
        foreach (var child in root.Children) {
            GetTreeNodes(child, ref nodes);
        }
    }
```

Код реализации алгоритма сохранения состояния игры представлен ниже:

```
public void SaveGame() {
    Vector3 positionPlayer = GameObject.FindGameObjectWithTag
(“Player”).transform.position;
    GameObject[] boxes = GameObject.FindGameObjectsWithTag (“Box”);
    List<Vector3> positionBoxes = new List<Vector3>();
    foreach (var box in boxes) {
       positionBoxes.Add(box.transform.position);
    }
    int curLvl = LevelBuilder.m_CurrentLevel;
    TreeNode child = new TreeNode(curLvl, TreeNodeFunctions.CurrentNode,
positionPlayer, positionBoxes);
    TreeNodeFunctions.CurrentNode.AddChild(child);
    TreeNodeFunctions.SetCurrentNode(child);
  }
```

Код реализации алгоритма загрузка игры представлен ниже:

```
public void LoadGame(Text text) {
    int loadId = int.Parse(text.text);
    TreeNode loadNode = TreeNodeFunctions.GetTreeNodeById(loadId);
    Vector3 loadPositionPlayer = loadNode.PositionPlayer;
    List<Vector3> loadPositionBoxes = loadNode.PositionBoxes.ToList();
    LevelBuilder.m_CurrentLevel = loadNode.Level;
    GameObject.FindGameObjectWithTag(“Player”).transform.position =
loadPositionPlayer;
    GameObject[] boxes = GameObject.FindGameObjectsWithTag (“Box”);
    for (int I = 0; I < boxes.Length; i++) {
       Destroy(boxes[i]);
    }
    for (int I = 0; I < loadPositionBoxes.Count; i++) {
       GameObject box = Instantiate(prefabBox, loadPositionBoxes[i],
Quaternion.identity);
       box.GetComponent<Box>().TestForOnCross();
    }
    TreeNodeFunctions.SetCurrentNode(loadNode);
  }
```

Код реализации алгоритма установки значений компонента сохраненного состояния в пользовательском интерфейсе представлен ниже:

```
private void SetSaveNodePrefab(GameObject instanceSaveNode, TreeNode
node) {
```

```
            Text text = instanceSaveNode.transform.Find("Text Id").
GetComponent<Text>();
            text.text = node.Id.ToString();
            int parentId = (node.Parent == null) ? -1 : node.Parent.Id;
            string textName = node.Name + "\n Parent: " + parentId;
            textName += (node.Parent == null) ? " (Root) " : "";
            instanceSaveNode.transform.Find("Text Name").
GetComponent<Text>().text = textName;
            Button btnLoad = instanceSaveNode.transform.Find("Button
load").GetComponent<Button>();
            btnLoad.onClick.AddListener(delegate {LoadGame(text);} );
            if (node.Id != 1) {
               Button btnDelete = instanceSaveNode.transform.Find("Button
Delete").GetComponent<Button>();
               btnDelete.onClick.AddListener(delegate { DeleteNode(node.Id); });
            } else {
               instanceSaveNode.transform.Find("Button
Delete").gameObject.SetActive(false);
            }
         }
```

Пример работы системы контроля версий состояний на основе реализованных выше алгоритмов. Остальные части кода для реализации других частей системы и игры в целом представлены в Приложении А.

Игра начинается с загрузки уровня. Пользователю представляется карта с ящиками, позициями крестов и персонажем. Также, в начале игры всегда создается запись начального состояния игры, соответствующая корневому узлу модели (рисунок 4.1). Далее, в процессе игры пользователь может с помощью кнопки «Load menu» открывать и закрывать соответствующее меню сохранений состояний, а используя кнопку «Save» делать сохранения состояний игры (рисунок 4.2). Когда пользователь нажимает в меню сохранений для конкретного сохранения кнопку «Load», то система загружает выбранное состояние (рисунки 4.3 – 4.4). Загруженное состояние становится текущим и все последующие сохранения становятся потомками уже нового загруженного состояния (до следующей загрузки другого состояния соответственно) (рисунок 4.5 – загрузили состояние №3 и сделали для него сохранение. В графе «parent» указано родительское

(предыдущее) сохранение состояния игры №3, а в графе «ID» указан порядковый номер выполненного сохранения состояния игры). Также, используя кнопку «Delete» пользователь может удалить какое-то конкретное состояние игры, и все его потомки станут потомками родителя удаляемого сохранения (рисунок 4.6 – удалили запись №3 и теперь все ее бывшие потомки стали иметь родителя №2, который был родителем записи №3).

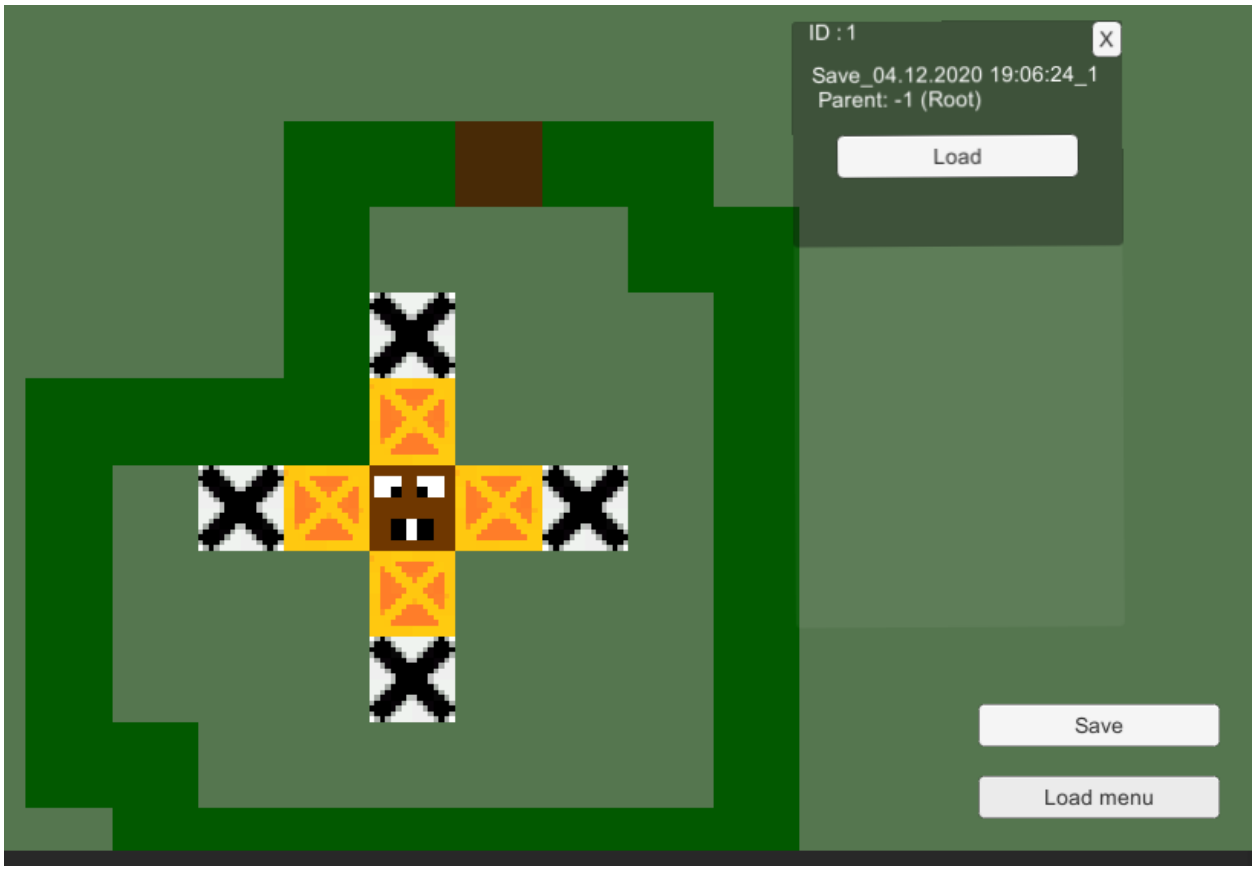


Рисунок 4.1 – Начальное состояние игры

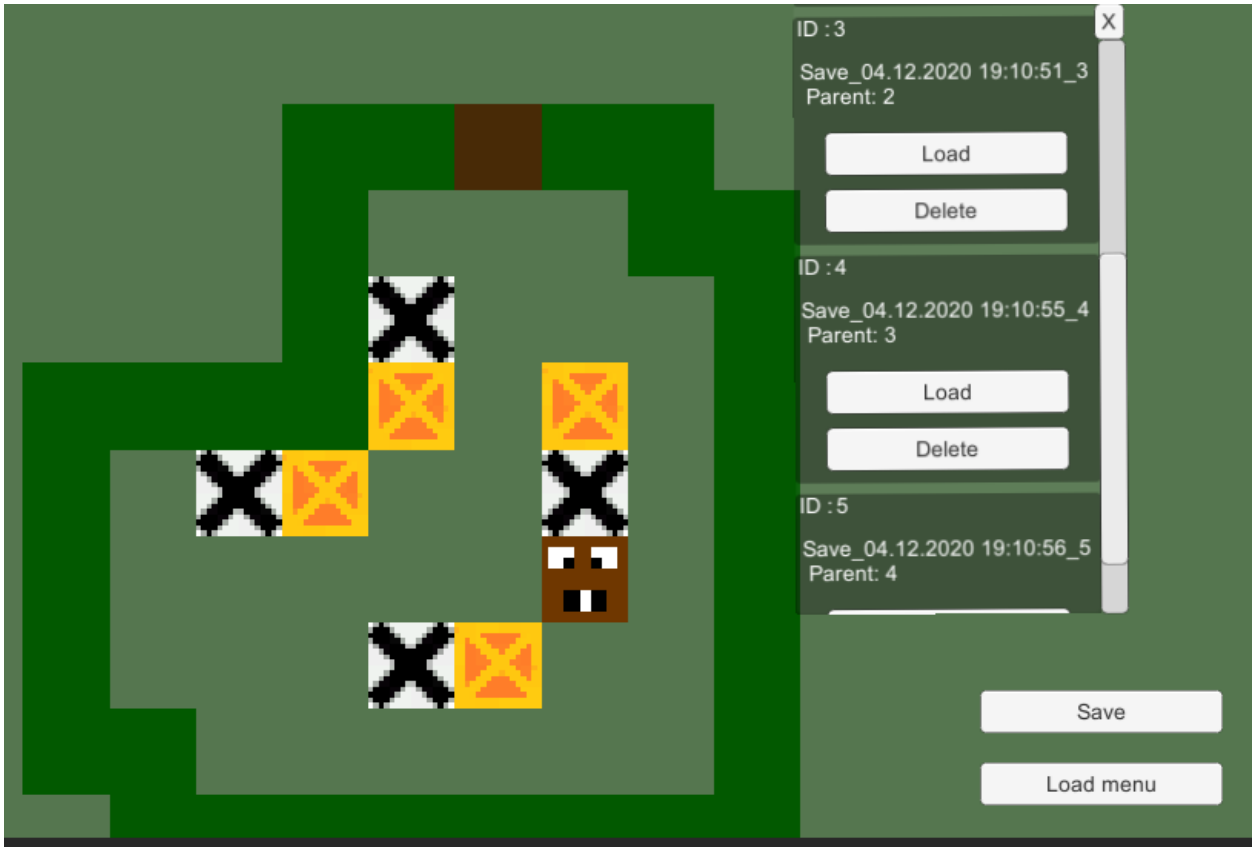


Рисунок 4.2 – Процесс игры (создание сохранений состояний)

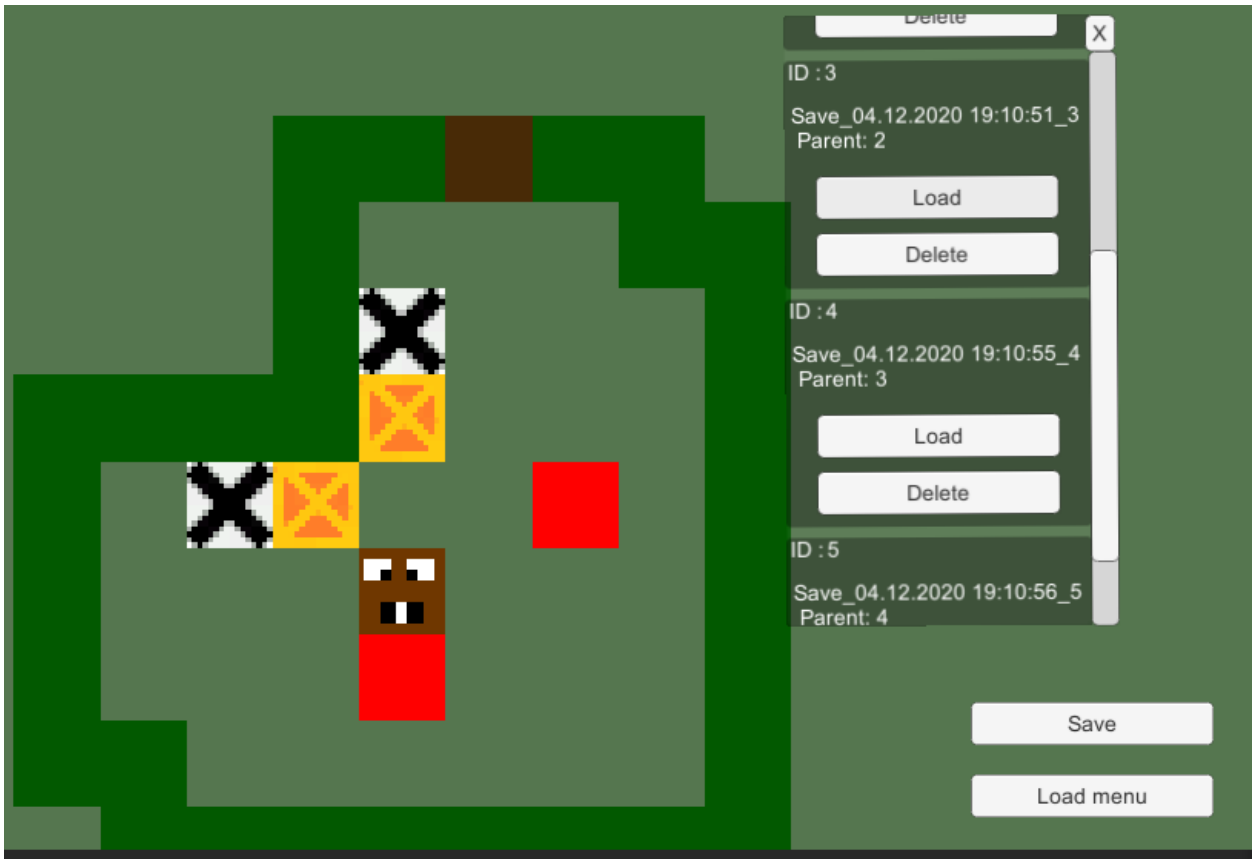


Рисунок 4.3 – Загрузка сохранения состояния игры №3

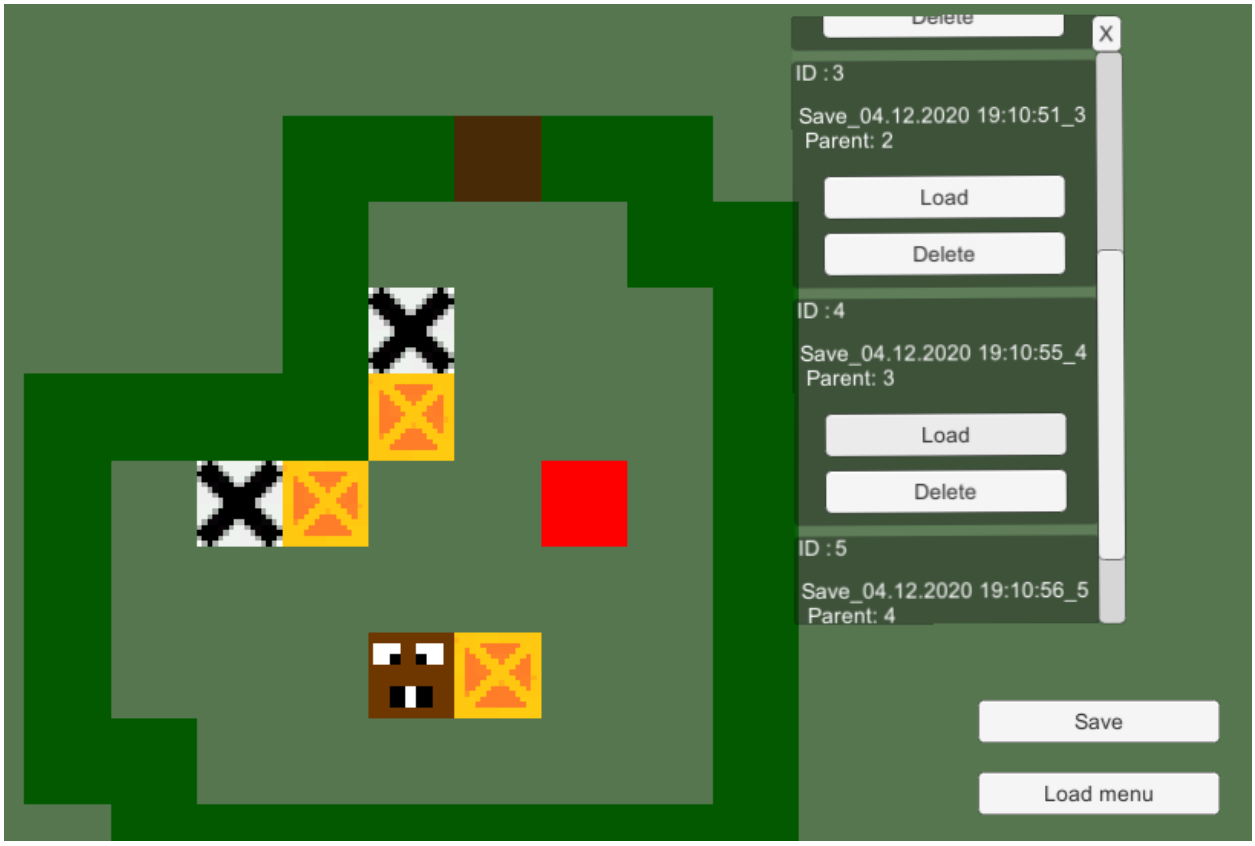


Рисунок 4.4 – Загрузка сохранения состояния игры №4

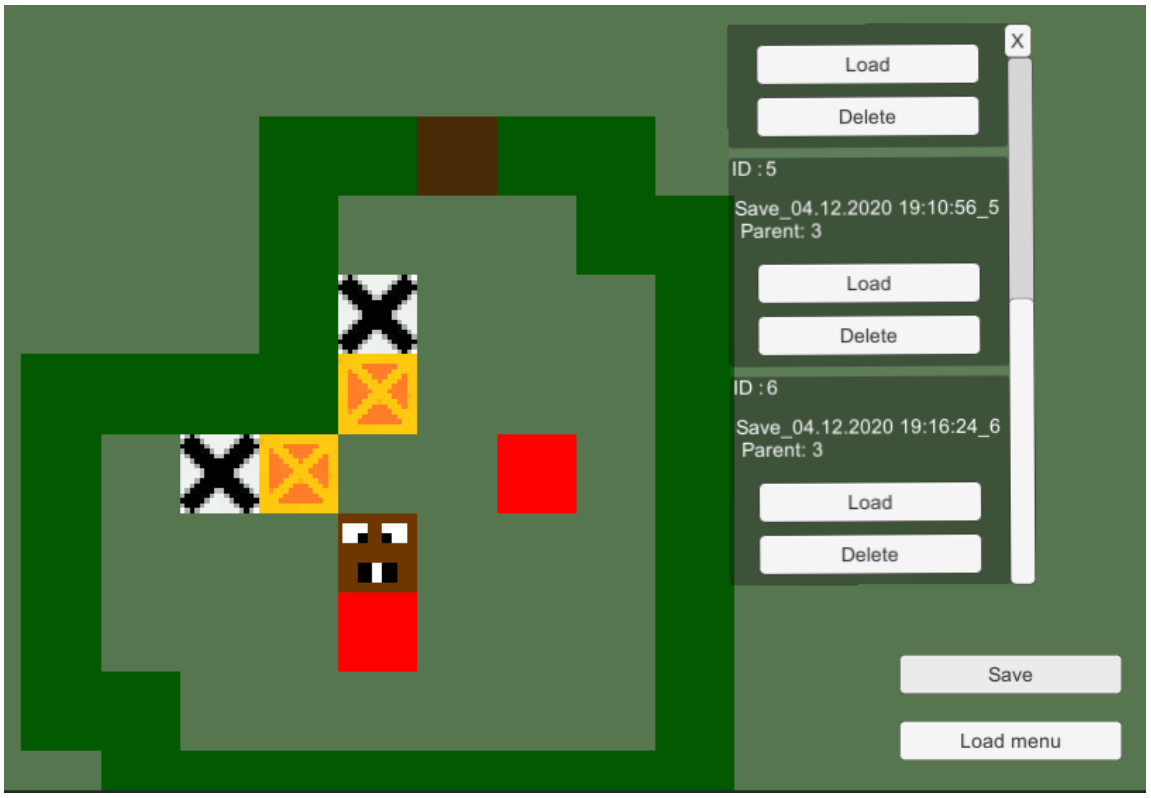


Рисунок 4.5 – Сохранение состояния игры для загруженного состояния игры №3

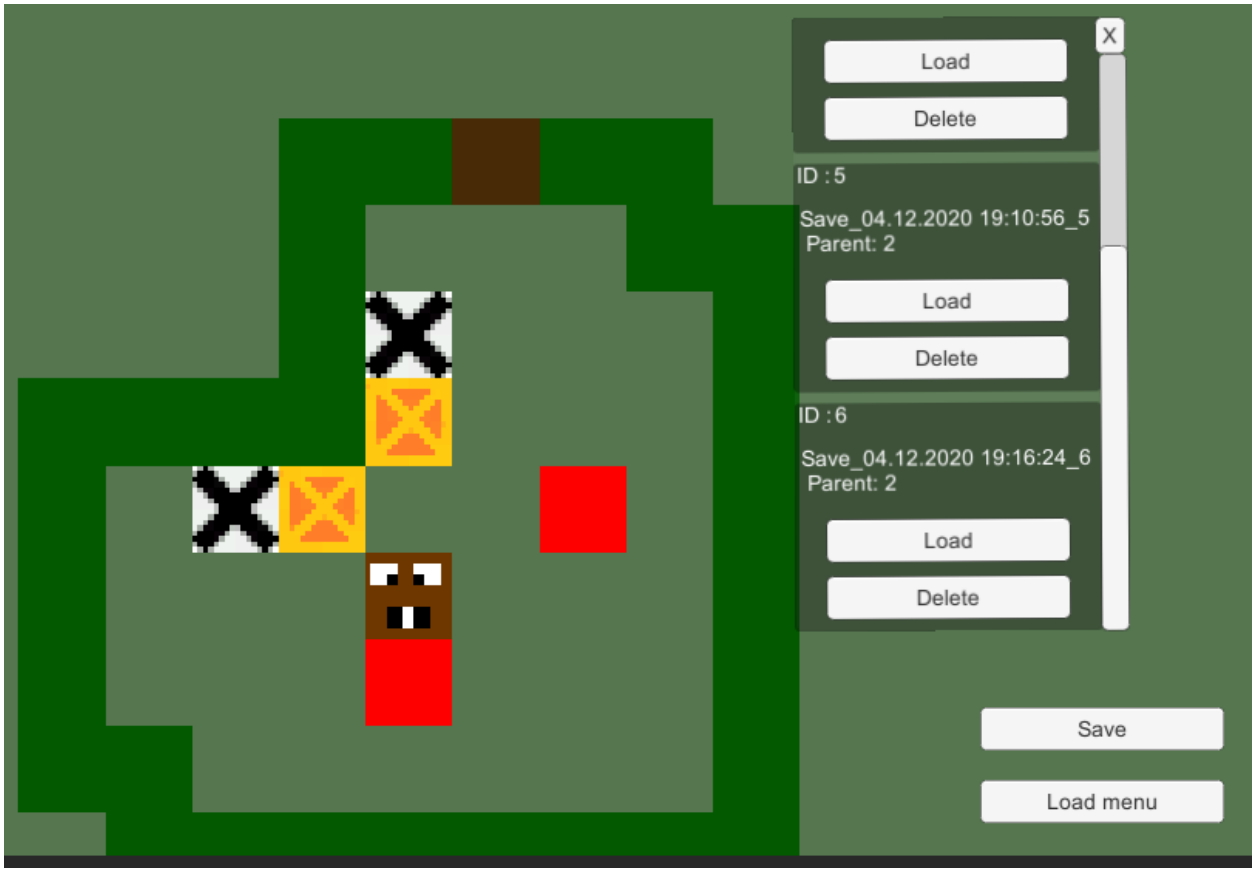


Рисунок 4.6 – Пример удаления сохранения состояния игры

5 ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Пользовательский интерфейс игры представлен на рисунке 5.1.

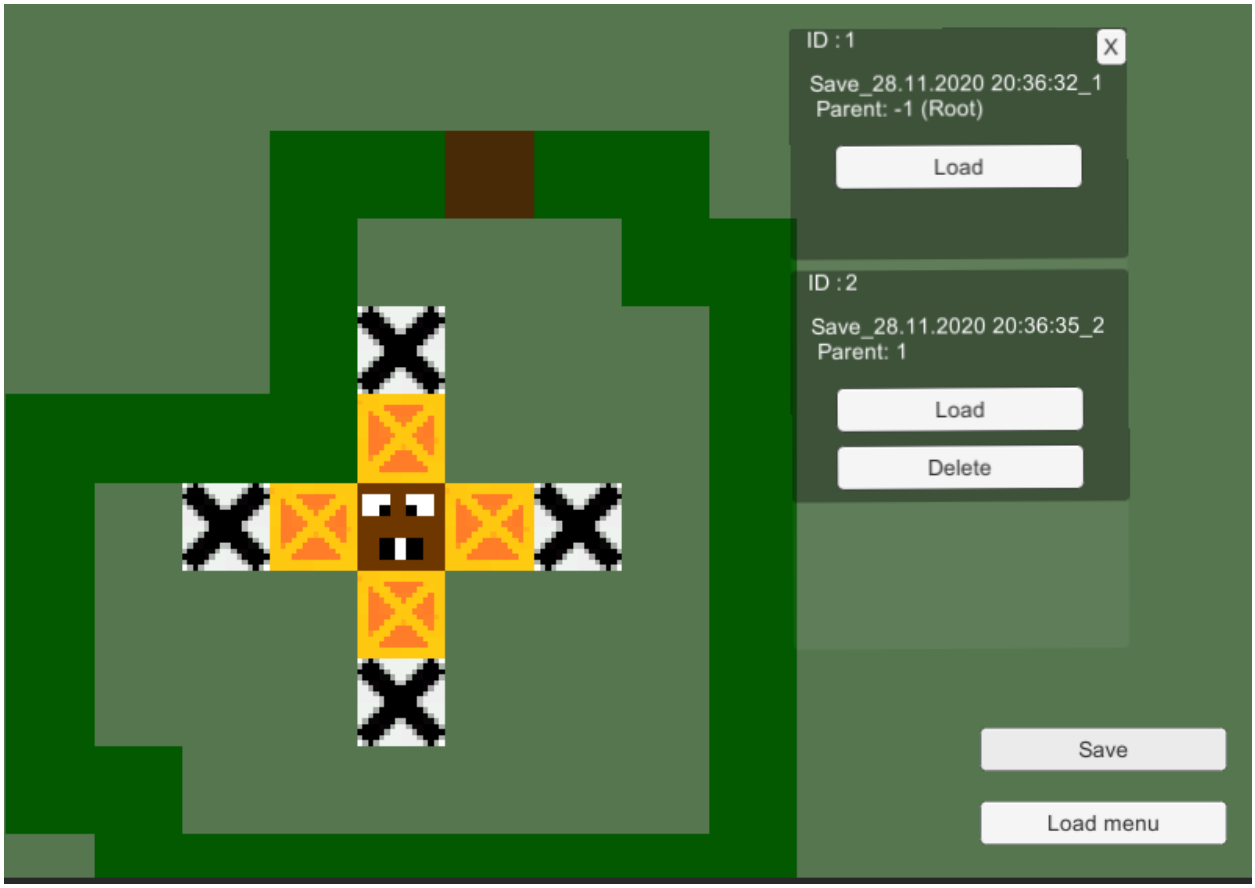


Рисунок 5.1 – Пользовательский интерфейс игры

На экране у пользователя представлены следующие элементы:

- Игровое поле с соответствующими элементами (стены, ящики, плитки с крестами, игрок);
- Кнопка «Save» – позволяет пользователю сохранить текущее состояние игры;
- Кнопка «Load menu» – при нажатии на эту кнопку пользователю открывается список со всеми ранее сделанными сохранениями;

Запись сохранения состоит из следующих элементов:

- Строка с порядковым номером записи;
- Строка с названием сохранения;

- Строка с порядковым номером родителя выбранной записи (если сохранение является корневым узлом, то после непосредственно номера идет приписка «Root»);
- Кнопка «Load» – позволяет пользователю загрузить выбранное сохранение;
- Кнопка «Delete» – позволяет пользователю удалить выбранное сохранение.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была описана предметная область и разработана модель для системы контроля версий, а также спроектированы и реализованы алгоритмы, основанные на разработанной модели, и описан пользовательский интерфейс.

Так как все задачи курсовой работы были выполнены, а требуемое программное обеспечение и требуемые алгоритмы были реализованы, курсовую работу можно считать выполненной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дикинсон, К. Оптимизация игр в Unity 5. Советы и методы оптимизации приложений [Текст] / К. Дикинсон. — ДМК-Пресс, 2017. — 306 с.

2. Кнут, Д. Э. Глава 2.3. Деревья // Искусство программирования = The Art of Computer Programming[Текст] / Д. Э. Кнут. — 3-е изд. — М.: Вильямс, 2002. — 720 с.

3. Шилдт, Г. C# 4.0: полное руководство [Текст] / Г. Шилдт. — ООО «И.Д. Вильямс», 2011. — 297 с.

ПРИЛОЖЕНИЕ А

(обязательное)

ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

«Box.cs»

```
using UnityEngine;

public class Box : MonoBehaviour
{
    #region Public properties

    public bool m_OnCross;  // True if box has been pushed on to a cross

    #endregion

    #region Public API
    // Avoid ability to move diagonally
    public bool Move(Vector2 direction)
    {
        if (BoxBlocked(transform.position, direction))
        {
            return false;
        }
        else
        {
            transform.Translate(direction); // Box not blocked, so move it
            TestForOnCross();
            return true;
        }
    }

    public void TestForOnCross()
    {
        GameObject[] crosses = GameObject.FindGameObjectsWithTag("Cross");

        foreach (var cross in crosses)
        {
            // On a cross
            if (transform.position.x == cross.transform.position.x
                && transform.position.y == cross.transform.position.y
                )
            {
                GetComponent<SpriteRenderer>().color = Color.red;
                m_OnCross = true;
                return;
            }
        }
        GetComponent<SpriteRenderer>().color = Color.white;
        m_OnCross = false;
    }

    #endregion

    #region Private API
    private bool BoxBlocked(Vector3 position, Vector2 direction)
    {
        Vector2 newPosition = new Vector2(position.x, position.y) + direction;
```

```
        GameObject[] walls = GameObject.FindGameObjectsWithTag("Wall");
        foreach (var wall in walls)
        {
            if (wall.transform.position.x == newPosition.x
                && wall.transform.position.y == newPosition.y
                )
            {
                return true;
            }
        }

        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
        foreach (var box in boxes)
        {
            if (box.transform.position.x == newPosition.x
                && box.transform.position.y == newPosition.y
                )
            {
                Box bx = box.GetComponent<Box>();
                if (bx && bx.Move(direction))
                {
                    return false;
                }
                else
                {
                    return true;
                }
            }
        }
        return false;
    }

    #endregion
}
```

«GameManager.cs»

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    #region Public fields

    public LevelBuilder m_LevelBuilder;
    public GameObject m_NextButton;
    public Player m_Player;

    #endregion

    #region Private fields

    private bool m_ReadyForInput;
    private Vector3 v3DoorPosition;
    private bool isDoorExists;

    #endregion

    #region Prefabs
```

```
    [SerializeField] private GameObject prefabDoor;

    #endregion

    #region Unity Actions

    private void Start()
    {
        m_NextButton.SetActive(false);
        ResetScene();
    }

    private void Update()
    {
        Vector2 moveInput = new Vector2(Input.GetAxisRaw("Horizontal"),
Input.GetAxisRaw("Vertical"));
        moveInput.Normalize();

        if (moveInput.sqrMagnitude > 0.5f)  // Button pressed or held
        {
            if (m_ReadyForInput)
            {
                m_ReadyForInput = false;
                m_Player.Move(moveInput);
                m_NextButton.SetActive(IsLevelComplete());
            }
        }
        else
        {
            m_ReadyForInput = true;
        }
    }

    #endregion

    #region Public API

    public void NextLevel()
    {
        m_NextButton.SetActive(false);
        m_LevelBuilder.NextLevel();
        m_LevelBuilder.Build();
        StartCoroutine(ResetSceneAsync());
    }

    public void ResetScene() => StartCoroutine(ResetSceneAsync());

    public bool IsLevelComplete()
    {
        Box[] boxes = FindObjectsOfType<Box>();
        foreach (var box in boxes)
        {
            if (!box.m_OnCross)
            {
                if (!isDoorExists)
                {
                    CloseDoor();
                }
                return false;
            }
        }
        if (isDoorExists)
```

```
        {
            OpenDoor();
        }
        return true;
    }

    #endregion

    #region Private API
    private IEnumerator ResetSceneAsync()
    {
        if (SceneManager.sceneCount > 1)
        {
            AsyncOperation asyncUnload = SceneManager.UnloadSceneAsync("LevelScene");
            while (!asyncUnload.isDone)
            {
                yield return null;
                //Debug.Log("Unloading...");
            }
            Debug.Log("Unload Done");
            Resources.UnloadUnusedAssets();
        }

        AsyncOperation asyncLoad = SceneManager.LoadSceneAsync("LevelScene",
LoadSceneMode.Additive);
        while (!asyncLoad.isDone)
        {
            yield return null;
            //Debug.Log("Loading...");
        }

        SceneManager.SetActiveScene(SceneManager.GetSceneByName("LevelScene"));
        m_LevelBuilder.Build();

        v3DoorPosition = GameObject.FindGameObjectWithTag("Door").transform.position;
        isDoorExists = true;

        m_Player = FindObjectOfType<Player>();
        Debug.Log("Level loaded");

        //Debug.Log("SaveRoot");
        TreeNodeFunctions.SaveRootNode();
    }

    private void OpenDoor()
    {
        GameObject door = GameObject.FindGameObjectWithTag("Door");
        Destroy(door.gameObject);
        isDoorExists = false;
    }

    private void CloseDoor()
    {
        Instantiate(prefabDoor, v3DoorPosition, Quaternion.identity);
        isDoorExists = true;
    }

    #endregion
}
```

«LevelBuilder.cs»

```
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class LevelElement       //defines each item in a level by mapping a single character
(i.e. #) to a prefab
{
    public string m_Character;
    public GameObject m_Prefab;
}

public class LevelBuilder : MonoBehaviour
{
    #region Public fields

    public static int m_CurrentLevel;
    public List<LevelElement> m_LevelElements;

    #endregion

    #region Private fields

    private Level m_Level;

    #endregion

    #region Public API
    public void NextLevel()
    {
        m_CurrentLevel++;
        if (m_CurrentLevel >= GetComponent<Levels>().m_Levels.Count)
        {
            m_CurrentLevel = 0;     // Wrap level back to 1st level
        }
    }

    public void Build()
    {
        m_Level = GetComponent<Levels>().m_Levels[m_CurrentLevel];

        // Offset coordinates so that centre of level is rougly at 0,0
        int startX = -m_Level.Width / 2;        // Save start x since needs to be reset in
loop
        int x = startX;
        int y = -m_Level.Height / 2;
        foreach (var row in m_Level.m_Rows)
        {
            foreach (var ch in row)
            {
                //Debug.Log(ch);
                GameObject prefab = GetPrefab(ch);

                if (prefab)
                {
                    //Debug.Log(prefab.name);
                    Instantiate(prefab, new Vector3(x, y, 0), Quaternion.identity);
                }
                x++;
            }
            y++;
            x = startX;
        }
```

```
    }

    #endregion

    #region Private API
    private GameObject GetPrefab(char c)
    {
        LevelElement levelElement = m_LevelElements.Find(le => le.m_Character ==
c.ToString());

        return levelElement != null ? levelElement.m_Prefab : null;
    }

    #endregion
}
```

«Levels.cs»

```
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class Level      // A single level
{
    #region Public fields

    public List<string> m_Rows = new List<string>();

    public int Height { get { return m_Rows.Count; } }
    public int Width  // length of longest row
    {
        get
        {
            int maxLength = 0;
            foreach (var r in m_Rows)
            {
                if (r.Length > maxLength)
                {
                    maxLength = r.Length;
                }
            }
            return maxLength;
        }
    }

    #endregion
}

public class Levels : MonoBehaviour
{
    #region Public fields

    public string fileName;
    public List<Level> m_Levels;

    #endregion

    #region UNITY ACTIONS
    private void Awake()
    {
        TextAsset textAsset = (TextAsset)Resources.Load(fileName);
```

```
        if (!textAsset)
        {
            Debug.Log("Levels: " + fileName + ".txt does not exists!");
            return;
        }
        else
        {
            Debug.Log("Levels imported");
        }

        string completeText = textAsset.text;
        string[] lines;
        lines = completeText.Split(new string[] { "\n" }, System.StringSplitOptions.None);

        m_Levels.Add(new Level());
        for (long i = 0; i < lines.LongLength; i++)
        {
            string line = lines[i];
            if (line.StartsWith(";"))
            {
                Debug.Log("New level added");
                m_Levels.Add(new Level());
                continue;
            }
            m_Levels[m_Levels.Count - 1].m_Rows.Add(line);      // Always adding level rows to
last level in list of levels
        }
    }

    #endregion
}
```

«LoadManager.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.UI;

public class LoadManager : MonoBehaviour
{
    #region Private fields

    [SerializeField] private GameObject prefabBox;
    [SerializeField] private GameObject prefabSaveNode;

    [SerializeField] private RectTransform content;

    [SerializeField] private TreeNodeFunctions _treeNodeFunctions;

    #endregion

    #region Public API
    public void LoadMenu()
    {
        List<TreeNode> nodes = _treeNodeFunctions.GetAllNodesInList();

        foreach (Transform child in content)
        {
            Destroy(child.gameObject);
```

```
        }

        foreach (TreeNode node in nodes)
        {
            var instance = Instantiate(prefabSaveNode.gameObject);
            instance.transform.SetParent(content, false);

            SetSaveNodePrefab(instance, node);
        }
    }

    public void LoadGame(Text text)
    {
        int loadId = int.Parse(text.text);
        TreeNode loadNode = TreeNodeFunctions.GetTreeNodeById(loadId);

        Vector3 loadPositionPlayer = loadNode.PositionPlayer;
        List<Vector3> loadPositionBoxes = loadNode.PositionBoxes.ToList();

        LevelBuilder.m_CurrentLevel = loadNode.Level;

        GameObject.FindGameObjectWithTag("Player").transform.position = loadPositionPlayer;

        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");

        for (int i = 0; i < boxes.Length; i++)
        {
            Destroy(boxes[i]);
        }

        for (int i = 0; i < loadPositionBoxes.Count; i++)
        {
            GameObject box = Instantiate(prefabBox, loadPositionBoxes[i],
Quaternion.identity);
            box.GetComponent<Box>().TestForOnCross();
        }

        TreeNodeFunctions.SetCurrentNode(loadNode);
    }

    #endregion

    #region Private API
    private void SetSaveNodePrefab(GameObject instanceSaveNode, TreeNode node)
    {
        Text text = instanceSaveNode.transform.Find("Text Id").GetComponent<Text>();
        text.text = node.Id.ToString();

        int parentId = (node.Parent == null) ? -1 : node.Parent.Id;
        string textName = node.Name + "\n Parent: " + parentId;
        textName += (node.Parent == null) ? " (Root) " : "";
        instanceSaveNode.transform.Find("Text Name").GetComponent<Text>().text = textName;

        Button btnLoad = instanceSaveNode.transform.Find("Button
load").GetComponent<Button>();
        btnLoad.onClick.AddListener(delegate {LoadGame(text);} );

        if (node.Id != 1)
        {
            Button btnDelete = instanceSaveNode.transform.Find("Button
Delete").GetComponent<Button>();
            btnDelete.onClick.AddListener(delegate { DeleteNode(node.Id); });
        }
```

```
        else
        {
            instanceSaveNode.transform.Find("Button Delete").gameObject.SetActive(false);
        }
    }

    private void DeleteNode(int idDeletedNode)
    {
        TreeNode deleteNode = TreeNodeFunctions.GetTreeNodeById(idDeletedNode);
        TreeNodeFunctions.DeleteNode(deleteNode);

        LoadMenu();
    }

    #endregion

}
```

«Player.cs»

```
using UnityEngine;

public class Player : MonoBehaviour
{
    #region Public API
    public bool Move(Vector2 direction)      // Avoid ability to move diagonally
    {
        if (Mathf.Abs(direction.x) < 0.5f)  // Will always set one of the coordinates to 0
        {
            direction.x = 0;
        }
        else
        {
            direction.y = 0;
        }

        direction.Normalize();      // Makes easier x or y = 1

        if (Blocked(transform.position, direction))
        {
            return false;
        }
        else
        {
            transform.Translate(direction);
            return true;
        }

    }

    public bool Blocked(Vector3 position, Vector2 direction)
    {
        Vector2 newPosition = new Vector2(position.x, position.y) + direction;

        GameObject[] walls = GameObject.FindGameObjectsWithTag("Wall");
        foreach (var wall in walls)
        {
            if (wall.transform.position.x == newPosition.x
                && wall.transform.position.y == newPosition.y
                )
            {
```

```
                return true;
            }
        }

        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
        foreach (var box in boxes)
        {
            if (box.transform.position.x == newPosition.x
                && box.transform.position.y == newPosition.y
                )
            {
                Box bx = box.GetComponent<Box>();
                if (bx && bx.Move(direction))
                {
                    return false;
                }
                else
                {
                    return true;
                }
            }
        }
        return false;
    }

    #endregion

}
```

«SaveManager.cs»

```
using System.Collections.Generic;
using UnityEngine;

public class SaveManager : MonoBehaviour
{
    #region Public API
    public void SaveGame()
    {
        Vector3 positionPlayer =
GameObject.FindGameObjectWithTag("Player").transform.position;
        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
        List<Vector3> positionBoxes = new List<Vector3>();

        foreach (var box in boxes)
        {
            positionBoxes.Add(box.transform.position);
        }

        int curLvl = LevelBuilder.m_CurrentLevel;
        TreeNode child = new TreeNode(curLvl, TreeNodeFunctions.CurrentNode, positionPlayer,
positionBoxes);
        TreeNodeFunctions.CurrentNode.AddChild(child);
        TreeNodeFunctions.SetCurrentNode(child);
    }

    #endregion
}
```

«TreeNodeFunctions.cs»

```
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class TreeNode
{
    #region Public fields
    public int Id { get; private set; }
    public string Name { get; private set; }
    public int Level { get; private set; }
    public Vector3 PositionPlayer { get; private set; }
    public List<Vector3> PositionBoxes { get; private set; }
    public TreeNode Parent { get; private set; }
    public List<TreeNode> Children { get; private set; } = new List<TreeNode>();

    #endregion

    #region Private static fields

    private static int count = 1;

    #endregion

    #region Constructors
    public TreeNode() { count++; }
    public TreeNode(int level, TreeNode parent, Vector3 positionPlayer, List<Vector3>
positionBoxes)
    {
        SetId(count);
        SetName();
        SetLevel(level);

        SetParent(parent);

        SetPositionPlayer(positionPlayer);
        SetPositionBoxes(positionBoxes);

        count++;
    }

    #endregion

    #region Add

    public void AddChild(TreeNode treeNode) => Children.Add(treeNode);

    #endregion

    #region Setters
    public void SetName() => Name = "Save_" + DateTime.Now + "_" + Id;
    public void SetId(int id) => Id = id;
    public void SetLevel(int level) => Level = level;
    public void SetParent(TreeNode parent) => Parent = parent;
    public void SetPositionPlayer(Vector3 positionPlayer) => PositionPlayer = positionPlayer;
    public void SetPositionBoxes(List<Vector3> positionBoxes) => PositionBoxes =
positionBoxes.ToList();

    #endregion

    #region Delete

    public void DeleteChild(TreeNode node) => Children.Remove(node);
```

```
    #endregion

    #region Prints
    public void PrintChildren()
    {
        foreach (var child in Children)
        {
            child.Print();
        }
    }

    public void PrintParent() => Debug.Log("Parent Id: " + Parent.Id);
    public void PrintCount() => Debug.Log("Count: " + count);

    public void Print()
    {
        Debug.Log("Id: " + Id);
    }

    #endregion
}

public class TreeNodeFunctions : MonoBehaviour
{
    #region Public fields
    public static TreeNode Root { get; private set; }
    public static TreeNode CurrentNode { get; private set; }

    #endregion

    #region Publuc static API
    public static void SetCurrentNode(TreeNode currentNode) => CurrentNode = currentNode;

    public static void SaveRootNode()
    {
        Vector3 positionPlayer =
GameObject.FindGameObjectWithTag("Player").transform.position;
        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
        List<Vector3> positionBoxes = new List<Vector3>();
        foreach (var box in boxes)
        {
            positionBoxes.Add(box.transform.position);
        }
        int curLvl = LevelBuilder.m_CurrentLevel;
        Root = new TreeNode(curLvl, null, positionPlayer, positionBoxes);

        SetCurrentNode(Root);
    }

    public static TreeNode GetTreeNodeById(int id) => GetTreeNodeByIdRecursive(Root, id);

    public static void DeleteNode(TreeNode node)
    {
        TreeNode parent = node.Parent;
        foreach (var child in node.Children)
        {
            parent.AddChild(child);
            child.SetParent(parent);
        }

        parent.DeleteChild(node);
    }
```

```
    #endregion

    #region Public API
    public List<TreeNode> GetAllNodesInList()
    {
        List<TreeNode> nodes = new List<TreeNode>();

        GetTreeNodes(Root, ref nodes);

        return nodes;
    }

    #endregion

    #region Private static API
    private static TreeNode GetTreeNodeByIdRecursive(TreeNode root, int id)
    {
        if (root.Id == id)
        {
            return root;
        }
        else
        {
            foreach (var child in root.Children)
            {
                TreeNode result = GetTreeNodeByIdRecursive(child, id);
                if (result != null)
                {
                    return result;
                }
            }
        }

        return null;
    }

    #endregion

    #region Private API
    private void GetTreeNodes(TreeNode root, ref List<TreeNode> nodes)
    {
        if (root != null)
        {
            nodes.Add(root);
        }

        foreach (var child in root.Children)
        {
            GetTreeNodes(child, ref nodes);
        }
    }
    #endregion
}
```