# Summary

## Вводная часть

В программе используется немного измененный подход MVC. Для упрощения и сокращения дополнительного кода был убран компонент controller, в данной архитектуре его роль совмещена с моделью.

Прежде чем говорить о плюсах и минусах и логики данного подхода, стоит обратить внимание на несколько важных фактов, которые нужно учитывать при разработке игры:

- оптимизация;
- возможность интегрировать или убрать сложную фичу в сжатые сроки;
- гибкость и прозрачность архитектуры;
- возможность переноса кода игры на другой движок или сервер;
- (баланс с пунктом ниже) уменьшить количество лишнего кода для реализации одной фичи - "программирование ради программирования" часто требует дополнительных временных затрат и усложняет проект;
- (баланс с пунктом выше) логика одной отдельной фичи в идеале должна быть зависеть от минимума других фичей - во избежания проблем “убрали слот в инвентаре, перестала ездить машина”;
- фичи должны быть максимально расширяемыми - должен быть некий уровень абстракций.

Таким образом архитектура должна включать в себя следующие пункты:

- минимизация использования наследников MonoBehaviour и

возможность кеширования;

- отделение логики от визуал;
- контроль инициализации и обновления программы;
- абстракции в разумных пределах для создания расширяемых шаблонов;
- (в идеале) базовая логика игры независима от движка.

Таким образом можно сформулировать некоторые правила архитектуры. **Model** основная единица содержащая данные и логику управления чего-либо. Может иметь визуальное представление, а может функционировать и без него.

**View** визуальное представление определенной модели в движке. Является наследником UnityEngine.Component, может использоваться для регистрации некоторых сообщений от движка.

**Component** (просто класс) конечная единица дополнительной, независимой логики или инструмент.

Применяя данную архитектуру можно выделить плюсы и минусы ее использования.

| Плюсы + | Минусы - |
|---|---|
| ● контролируемый Update;<br>● подавляющее большинство классов не являются наследниками типа UnityEngine.Component;<br>● простота отладки; | ● двухсторонняя связь, model знает про свой view, а view про свою model;<br>● иногда нужно писать чуть больше кода, чем в случае работы с monobehaviour; |

| | |
|---|---|
| ● разделенная логика графики и визула;<br>● ясность и простота архитектурных шаблонов;<br>● легко добавлять фичи и расширять функционал;<br>● при желании можно безвредно убрать из моделей типы UnityEngine и отвязать всю логику от Unity. | ● необходимо понимание языка на среднем уровне. |

## Описание архитектуры программы

У программы есть единственная точка входа **GameCore**, наследник класса **MonoBehaviour**, у данного класса имеется единственный главный метод Update, который расходится по всем моделям приложения.

GameCore содержит в себе основную модель **MainModel**, которая имеет список *singleton* моделей с общей логикой работы приложения, а также инициализирует модели в данном списке.

Среди прочих MainModel содержит ссылку на **SimulationModel**. Это модель отвечающая за всю логику связанную с симуляцией. **SimulationModel** хранит еще список синглтонт моделей, которые обеспечивают работу симуляции.

SimulationModel, инициализирует все необходимые для симуляции модели, управляет их обновлением, и содержит в себе логику управления

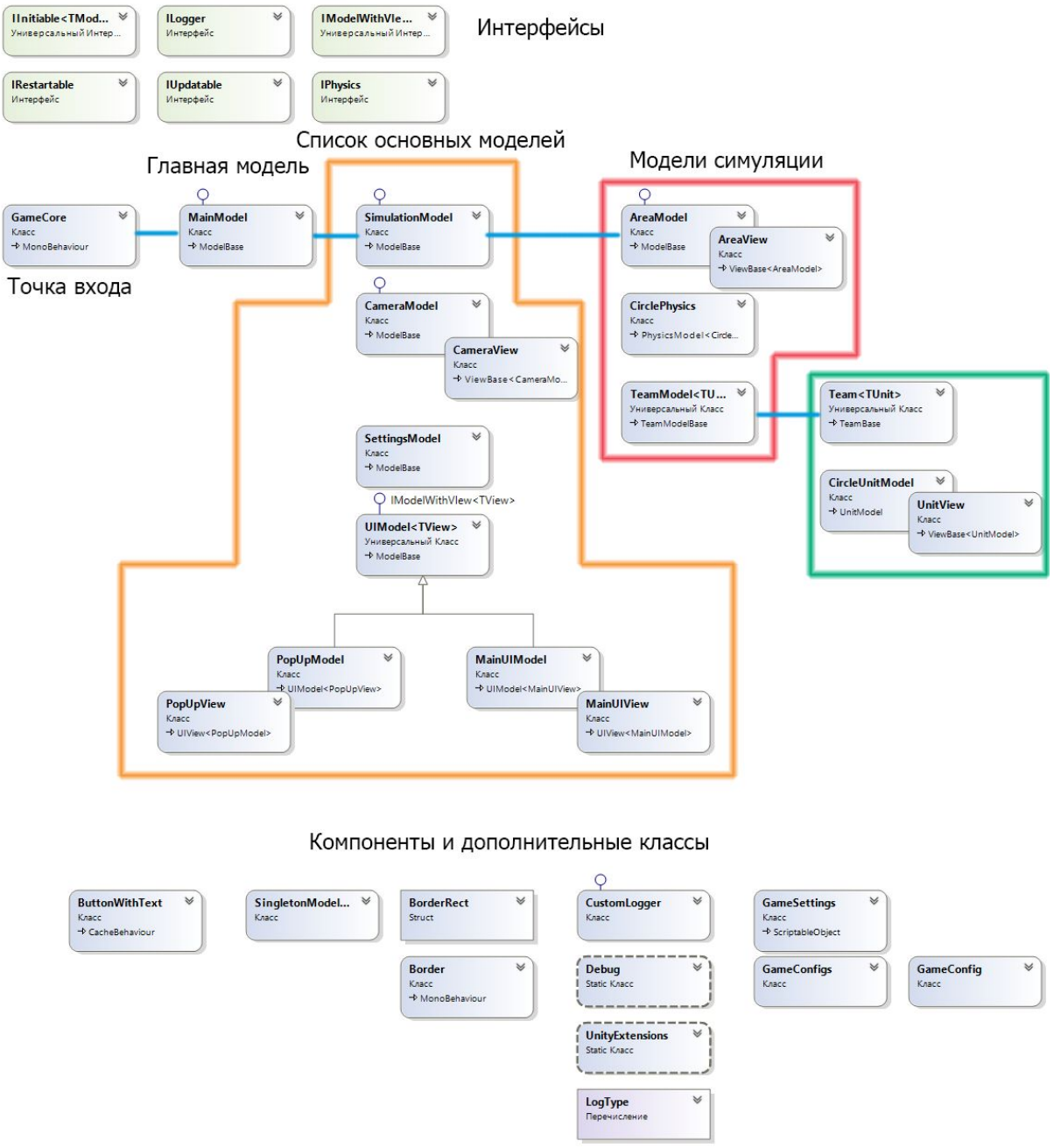
симуляцией.

Стоит отметить, что через **GameCore** можно получить доступ ко всем моделям из любого места в коде. Сам GameCore является неуничтожаемым (*dont destroy on load*) сингдтоном. Через этот же объект можно регулировать скорость симуляции.

Если какая-то модель представляет собой объект, который должен быть визуализирован и представлен на сцене в Unity, то у этой модели должна быть ссылка на типа наследуемый от **ViewBase**. **ViewBase** является наследником **CachedBehaviour**, классом, который кеширует и перезаписывает transform, для более оптимизированного обращения к компоненту.

Ниже приведу схему классов (ее можно посмотреть по ссылке, а также она загружена в гит).

Дублирую ссылку на репозиторий.


IInitiable<TMod...
Универсальный Интер...
ILogger
Интерфейс
IModelWithVIe...
Универсальный Интер...
Интерфейсы
IRestartable
Интерфейс
IUpdatable
Интерфейс
IPhysics
Интерфейс
Список основных моделей
Главная модель
Модели симуляции
GameCore
Класс
MonoBehaviour
MainModel
Класс
ModelBase
SimulationModel
Класс
ModelBase
AreaModel
Класс
ModelBase
AreaView
Класс
ViewBase<AreaModel>
Точка входа
CameraModel
Класс
ModelBase
CameraView
Класс
ViewBase<CameraMo...
CirclePhysics
Класс
PhysicsModel<Circle...
TeamModel<TU...
Универсальный Класс
TeamModelBase
Team<TUnit>
Универсальный Класс
TeamBase
SettingsModel
Класс
ModelBase
CircleUnitModel
Класс
UnitModel
UnitView
Класс
ViewBase<UnitModel>
IModelWithVIew<TView>
UIModel<TView>
Универсальный Класс
ModelBase
PopUpModel
Класс
UIModel<PopUpView>
MainUIModel
Класс
UIModel<MainUIView>
PopUpView
Класс
UIView<PopUpModel>
MainUIView
Класс
UIView<MainUIModel>


Компоненты и дополнительные классы
ButtonWithText
Класс
CacheBehaviour
SingletonModel...
Класс
BorderRect
Struct
Border
Класс
MonoBehaviour
CustomLogger
Класс
Debug
Static Класс
UnityExtensions
Static Класс
LogType
Перечисление
GameSettings
Класс
ScriptableObject
GameConfigs
Класс
GameConfig
Класс