# Оглавление

[Оглавление 5](#_Toc358568989)

[Введение 7](#_Toc358568990)

[1. Аналитический раздел 8](#_Toc358568991)

[1.1. Постановка задачи 8](#_Toc358568992)

[1.2. Анализ алгоритмов определения частоты основного тона 8](#_Toc358568993)

[1.2.1. Алгоритмы, работающие во временном пространстве 9](#_Toc358568994)

[1.2.2. Алгоритмы, работающие в частотном пространстве 11](#_Toc358568995)

[1.3. Анализ методов и алгоритмов синхронизации нот 12](#_Toc358568996)

[1.3.1. Динамическое программирование 13](#_Toc358568997)

[1.3.2. Планирование в реальном времени 13](#_Toc358568998)

[1.3.3. Функция плотности вероятности 14](#_Toc358568999)

[1.3.4. Динамическая трансформация времени 14](#_Toc358569000)

[1.3.5. Скрытые Марковские модели 17](#_Toc358569001)

[1.4. Обзор существующих решений синхронизации нот 19](#_Toc358569002)

[1.4.1. Tonara 19](#_Toc358569003)

[1.5. Выводы по аналитическому разделу 20](#_Toc358569004)

[2. Конструкторская часть 21](#_Toc358569005)

[2.1. Модифицированный алгоритм определения ЧОТ 21](#_Toc358569006)

[2.1. Разработка оптимизированной СММ. 23](#_Toc358569007)

[2.2. Равномернотемперированный музыкальный строй 24](#_Toc358569008)

[2.3. Представление нот и пауз в матрице переходов 25](#_Toc358569009)

[2.4. Алгоритм Витерби в реальном времени. 25](#_Toc358569010)

[2.5. Синхронизация музыканта и системы в начале. 27](#_Toc358569011)

[2.6. Расчет темпа в реальном времени. 27](#_Toc358569012)

[2.7. Методика оценки степени синхронизированности. 30](#_Toc358569013)

[2.8. Выводы по конструкторскому разделу 34](#_Toc358569014)

[3. Технологическая часть 35](#_Toc358569015)

[3.1. Требования к вычислительной системе 35](#_Toc358569016)

[3.2. Выбор языка и среды программирования 35](#_Toc358569017)

[3.2.1. Трансляция C# в JavaScript 36](#_Toc358569018)

[3.3. Структура разрабатываемой системы 38](#_Toc358569019)

[3.3.1. Модуль определения частоты основного тона 38](#_Toc358569020)

[3.3.2. Модуль для обработки MusicXML файлов 40](#_Toc358569021)

[3.3.3. Модуль для отрисовки нот 41](#_Toc358569022)

[3.3.4. Модуль для обработки аудио в офлайн и онлайн режимах 42](#_Toc358569023)

[3.3.5. Модуль для анализа нотных записей. 43](#_Toc358569024)

[3.4. Пользовательский интерфейс 44](#_Toc358569025)

[3.5. Описание форматов файлов 45](#_Toc358569026)

[3.5.1. Midi 45](#_Toc358569027)

[3.5.2. MusicXML 45](#_Toc358569028)

[3.6. Используемые сторонние библиотеки 46](#_Toc358569029)

[3.6.1. NAudio 46](#_Toc358569030)

[4. Экспериментально-исследовательский раздел 47](#_Toc358569031)

[4.1. Тестирование алгоритмов определения ЧОТ 47](#_Toc358569032)

[4.2. Тестирование алгоритма Витерби в реальном времени. 49](#_Toc358569033)

[4.3. Тестирование методики оценки степени синхронизированности. 50](#_Toc358569034)

[4.4. Результаты 53](#_Toc358569035)

[4.5. Анализ результатов тестирования 55](#_Toc358569036)

[Заключение 59](#_Toc358569037)

[Список использованных источников 60](#_Toc358569038)

[Приложение А 62](#_Toc358569039)

[Тестовые мелодии 62](#_Toc358569040)

[Приложение Б 64](#_Toc358569041)

[Вероятности СММ на примере Twinkle Twinkle Little Star 64](#_Toc358569042)

[Приложение В 67](#_Toc358569043)

[Приложение Г 68](#_Toc358569044)

# Введение

С давних пор известно, что занятие музыкой, в частности игра на музыкальных инструментах, развивает интеллект, слух, творческие способности человека и другие полезные навыки. В связи с этим, было бы хорошо, если каждый человек развивал свои музыкальные способности, в частности, умел играть на каком-нибудь музыкальном инструменте. Некоторые обучаются этому навыку в музыкальной школе еще в детстве, но у большинства уже нет времени на музыкальные заведения. Кроме того, может оказаться, что обучение у репетитора стоит не малых материальных затрат.

Примерно в конце 20-го века встал вопрос о самостоятельном или автоматизированном обучении игре на музыкальных инструментах. Появились различные научные дисциплины о музыке и программное обеспечение для развития слуха, для обучения нотным записям, для развития техники игры на различных музыкальных инструментах*, для синхронизации исполнения композиции музыкантом и нот*.

Существующие системы синхронизации в реальном времени способны функционировать на высокопроизводительных ЭВМ. К тому же они не могут оценивать насколько точно музыкант сыграл музыкальную композицию с точки зрения попадания в ноты и темпа. Разработке методики оценки степени синхронизации нотных записей и живой музыки и посвящена данная магистерская диссертация.

# Аналитический раздел

## Постановка задачи

Целью данного дипломного проекта является разработка программного обеспечения по обработке цифровых аудио сигналов с целью дальнейшей синхронизации и оценки нотных записей в реальном времени.

Для достижения данной цели необходимо решить следующие задачи:

* Анализ алгоритмов определения частоты основного тона.
* Анализ алгоритмов синхронизации временных рядов.
* Разработка алгоритма синхронизации нотных записей в реальном времени.
* Разработка алгоритма оценивания темпа исполнения.
* Разработка методики оценки степени синхронизации.
* Реализация программного комплекса.
* Исследование разработанных алгоритмов и методики.

## Анализ алгоритмов определения частоты основного тона

Алгоритм определения частоты основного тона – это алгоритм для оценки частоты основного тона квазипериодического сигнала, используемый обычно для цифровой записи речи или музыкальной ноты. На протяжении многих лет ведутся исследования для поиска оптимального алгоритма. Основой проблемой является то, что в обрабатываемом звуковом фрагменте существуют различные шумы и обертоны, имеющие определенные частоты, которые мешают определить частоту основного тона. Как видно на рисунке ниже, если раскладывать простой синусоидальный сигнал в спектр Фурье и выбирать частоту с наибольшей амплитудой, то это и будет частота основного тона. Но если сигнал будет иметь дополнительные гармоники, то такой простой алгоритм, который подходил бы для первого случая, сработает неправильно. Ниже будут описаны некоторые из алгоритмов определения частот основного тона.

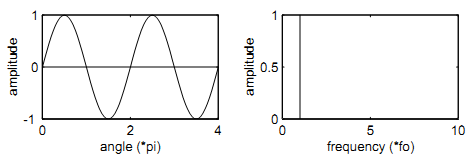


Рисунок 1 - Правильное определение частоты синусоидального сигнала.

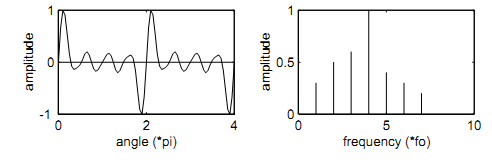


Рисунок 2 - Ошибочное определение частоты более сложного сигнала.

По возможности определения нескольких основных частот в звуковом сигнале, методы подразделяются на:

* Монофонические
* Полифонические

Из-за высокой сложности и большой неточности, в данном проекте анализируются и реализуются только монофонические алгоритмы, однако стоит заметить, что *нейронные сети* хорошо подходят для задачи определения нескольких частот в сигнале.

По типу входных данных эти алгоритмы подразделяются на:

* Алгоритмы, работающие во временном пространстве
* Алгоритмы, работающие в частотном пространстве

### Алгоритмы, работающие во временном пространстве

##### Пересечение с нулем (Zero crossing)

Данный алгоритм является самым простым и часто используемым, особенно в электронике, математике и обработке изображений[1)]. В нем подсчитывается количество пересечений сигнала с осью времени (точки, где сигнал меняет знак), а затем происходит вычисление частоты с помощью формулы:

 (1.4)

В которой *st* – амплитуда сигнала в момент времени t, – индикаторная функция, возвращающая 1, если аргумент принимает значение true, и 0 – в противоположном случае.

Существенным недостатком данного метода является то, что в случае если сигнал содержит высокочастотные спектральные компоненты, то частота основного тона может определиться неправильно. Пример сигнала, на котором произойдет неверное определение частоты, показан на Рисунок 1.7 справа: там количество пересечений будет 3, а не 2, как на Рисунок 1.7 слева.

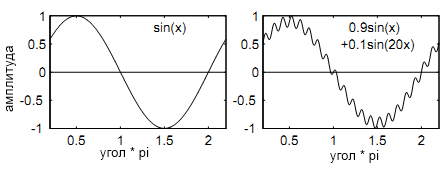


Рисунок 3 - Пример, на котором алгоритм пересечения с нулем дает неверный результат (справа).

##### Алгоритм автокорелляции

В данном алгоритме ищется сходство между сигналом и его сдвинутой копией[5)]. Математическое определение автокорелляционной функции:

 (1.5)

В которой *n* – коэффициент сдвига. Если исходный сигнал периодический, то и автокорелляционная формула будет периодической и период основного тона, будет определяться как время между нулем и первым минимумом в автокорелляционной функции.

Недостатком данного метода является то, что он может работать только для ограниченного диапазона частот (низких и высоких).

Существуют метод, основанный на автокорелляции, который называется YIN. Он работает с большим диапазоном частот и дает очень точный результат, однако имеет большую вычислительную сложность. Поэтому данный алгоритм не был выбран в данном проекте.

### Алгоритмы, работающие в частотном пространстве

#### Преобразование фурье

Для того, чтобы иметь возможность проводить операции над частотным спектром звукового сигнала, его нужно разложить в ряд Фурье.

#### Гармоническое перемножение спектров

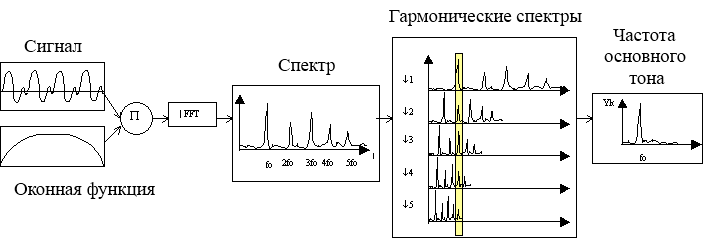


Рисунок 4 - Этапы работы алгоритма гармонического перемножения спектров.

Если входной сигнал является музыкальной нотой, тогда его спектр будет состоять из набора пиков, соответствующих частотам основного тона, умноженных на целые коэффициенты, как это представлено на Рисунок 1.8. Таким образом, если сжать спектр несколько раз и сравнить с оригинальным спектром, то можно заметить, что первый максимум в исходном спектре совпадает со вторым максимумом в спектре, сжатом в два раза и совпадает с третьим максимумом в спектре, сжатом в три раза. Далее, если эти спектры перемножить, то получится результирующий спектр с одним максимумом, соответствующим частоте основного тона сигнала, как это показано на рисунке справа.

Несмотря на то, что алгоритм гармонического умножения спектров имеет невысокую вычислительную сложность, он может вычислять частоты только в ограниченном диапазоне, который напрямую зависит от размера исходного семпла. Т.к. в данном проекте используются короткие семплы, исходя из требований работы в реальном режиме, данный алгоритм не был выбран.

## Анализ методов и алгоритмов синхронизации нот

Исследования в области синхронизации нот и компьютерно-человеческого взаимодействия были впервые проделаны в 1984.

Первоначальные алгоритмы динамического программирования были использованы для сопоставления паттернов из ожидаемых нот и реально играемых музыкантом нот.

Вероятностные модели были впервые введены в 1990. Грабб и Денненберг (1998) использовали функцию распределения вероятностей для определения наиболее вероятно играемой позиции в нотах.

Исследовательская группа из IRCAM института провела много исследований в области синхронизации нот, использовав достижения из биологических алгоритмов сравнивания цепочек и обработки речи. Орио и Шварц (2001) использовали алгоритм динамической трансформации времени (DTW) для анализа аудио сигналов и выравнивания их к музыкальным нотам.

В недавних исследованиях в IRCAM были использованы скрытые Марковские сети (СММ) для моделирования нот и возможных отклонений от идеальной игры, среди которых были неправильно сыгранная нота, пропущенная нота или неправильно добавленная нота. В скрытых Марковских моделях используется моделирование вероятностей переходов для каждого события.

Рафаэль (1999) разработал очень похожую работу. Он выбрал СММ как предпочитаемый метод, с помощью которого моделируется и отслеживаются ноты определенного участка, хотя он также начинал с объединения Байесовских вероятностей.

### Динамическое программирование

Динамическое программирование для цели синхронизирования нот впервые было введено Барри Версое в 1984.

Основная методология Версое и Данненберга состоит в сравнении того, что было сыграно музыкантом и ожидаемой последовательности нот. Таким образом, рекурсивный алгоритм строил оптимальный путь, который музыкант проделывал по нотам.

Версое (1984) выделил три стадии обработки для синхронизации нот: прослушивание, исполнение и обучение.

### Планирование в реальном времени

Данненберг (1989) ввел разделение между *реальным* и *виртуальным* временем. Реальное время - это физическое время исполнения, в то время как виртуальное время – внутренняя мера времени. Данненберг разделял две этих концепции с целью отслеживания относительного прогресса музыканта в нотах.

### Функция плотности вероятности

В 1989 Данненберг и его студент Лорин Грабб написали о новом достижении в отслеживании нот, основанное на функции плотности вероятности.

Из предыдущей позиции в нотах, программа непрерывно оценивает расстояние от предыдущей позиции. Затем алгоритм использует наиболее недавние наблюдения частот, спектральных характеристик, изменений амплитуды и другого для определения ноты, которой музыкант достиг в данный момент.

### Динамическая трансформация времени

IRCAM представил два исследовательских проекта на 2001 Международной конференции компьютерной музыки: один заключался в использовании алгоритма DTW (динамической трансформации времени), а другой заключался в использовании Скрытых Марковских Сетей (СММ).

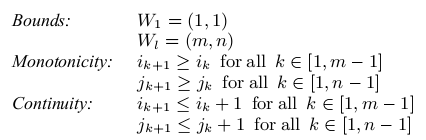
DTW алгоритм ищет наилучшее сопостовление между двумя последовательностями, анализируя меру схожести между полученным и ожидаемым аудио сигналами и выравнивая позиции нот.

Дурбин (1998) представил методику, основанную на Скрытых Марковских Сетях для сопоставления последовательностей. Орио и Шварц проанализировали методы DTW и СММ и выяснили, что в целом они аналогичны за тем исключением, что DTW является более оптимальным по использованию памяти для больших файлов. Тем не менее, более поздние работы IRCAM по синхронизации нот были сконцентрированы целиком на скрытых марковских моделях, в то время как DTW почти не рассматривался далее.

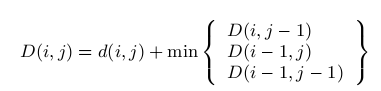
Алгоритм динамической трансформации шкалы времени находит оптимальное выравнивание двух временных последовательностей, но этого не достаточно для приложений в реальном времени, так как он требует сразу две цепочки последовательностей.

DTW алгоритм сопоставляет временные последовательности и , отыскивая минимальный по стоимости путь , где каждое – упорядоченная пара , такая что означает что точки и являются выровненными. Выравнивание определяется с помощью функции локальных стоимостей , которая обычно представляется в виде матрицы, которая назначает соответствующий вес для выравнивания каждой паре . Стоимость является нулевой для идеального совпадения и положительной в другом случае. Стоимость пути – это сумма совпадений локальных весов на продолжении пути.

Несколько ограничений накладываются на , которая обозначает путь, ограниченный концами двух последовательностей. Он является продолженным. Формально это выглядит так:



Минимальная стоимость пути может быть рассчитана за квадратичное время с использованием динамического программирования, используя рекурсию:



В этой формуле – стоимость минимального пути из в и . Таким образом, путь может быть получен с помощью обратной рекурсии из .

Тем не менее, в случае выравнивания в реальном времени с частично неизвестными последовательностями, несколько изменений должны быть внесены в DTW алгоритм. В стандартном DTW, длины последовательностей предоставляют одно из граничных условий поиска. В случае реального времени, это условие должно быть оценено вдоль оптимального пути. Вследствие этого, диагонали матрицы стоимостей неизвестны, таким образом, ограничения глобального пути не могут быть реализованы напрямую. Другое изменение которое нужно привнести состоит в том, чтобы минимальный по стоимости путь должен быть рассчитан только в прямом направлении. Далее, для того чтобы запустить в реальном времени с продолжительными длинными последовательностями, полный алгоритм должен быть линеен по всей длине последовательности, таким образом инкрементальный шаг должен быть постоянным.

Допустим что частично неизвестная последовательность. Тогда для каждого (которое измеряется в целых единицах, соответствующих индексам ), нужно найти лучшее выравнивание последовательности к другой начальной подпоследовательности . Данное решение удовлетворяет условием выше и алгоритму DTW в офлайн режиме. Переменные и указывают на текущие позиции в последовательностях и соответственно, которые инициализировались точкой старта каждой серии.

Главный цикл алгоритма вычисляет некоторые строки и столбцы пути матрицы стоимостей. В вычислениях обычно задействована стандартная рекурсивная формула DTW алгоритма, ограниченная значениями матрицы, которые уже были вычислены. Стоимость пути нормализуется длинной пути, таким образом, длины меняющихся длин могут быть сравнены в функции . Количество рассчитанных ячеек определяются параметром . Если новая строка становится вычисленной, номер ряда инкрементируется и ячейки в последних колонках, включая, рассчитываются.

### Скрытые Марковские модели

Отслеживание нот также может быть реализовано с помощью скрытых марковских моделей (СММ).

Скрытые Марковские модели – это стохастический инструмент, используемый во многих областях для распознавания речи, сопоставления биологических цепочек. Системы реального времени, в которых генерируется какой-либо наблюдаемый сигнал, могут быть смоделированы с помощью скрытых Марковских моделей. В отдельных случаях к таким системам можно включить и недетерминированные системы, т.е. такие системы, чье поведение не может быть предсказано точно с использованием множества алгоритмических правил или формул.

Вероятности используются в скрытых Марковских сетях для описания наблюдаемого поведения системы и внутренних (скрытых) состояний системы. СММ затем может использовать для обработки этих наблюдаемых сигналов для объяснения поведения системы и расчета вероятностных оценок о будущем поведении системы.

Система, смоделированная с помощью СММ, может рассматриваться как определенное количество состояния в любое время. Можно извлечь информацию о том, в каком состоянии находится система с помощью анализа последних выходов системы (наблюдений). Действительные состояния сами по себе не могут быть обнаружены просто с помощью последовательности наблюдений как результат, пришедший из системы. Наблюдаемый выход может быть интерпретирован как «вероятностная функция нахождения системы в состоянии».

Связь между индивидуальными состояниями и наблюдениями не является связью «многие ко многим». Одно наблюдение может быть выведено из многих состояний системы. В то время как существует более чем одно возможное наблюдение в. Фундаментальная разница между скрытыми марковскими моделями и марковскими цепочками заключается в том, что это соотношение не один к одному между состояниями и наблюдениями в скрытой марковской модели, в отличие от Марковских цепочек.

#### 1.2.5.1. Отслеживания нот с помощью скрытых марковских моделей

Музыкальные ноты могут представляться с помощью последовательности музыкальных событий.

Целью является поиск наиболее вероятной последовательности состояний, которая может быть сгенерирована из последовательности наблюдений, полученных от игры музыканта.

Алгоритм Витерби может использоваться для поиска наиболее вероятного пути среди состояний скрытой марковской модели.

Реализованный стандартным образом, это т алгоритм ищет глобальный оптимальный путь среди состояний скрытой марковской модели для определения наиболее вероятного текущего состояния, используя историю наблюдений. Однако в моделировании нот требуется локальный оптимальный путь до текущей позиции.

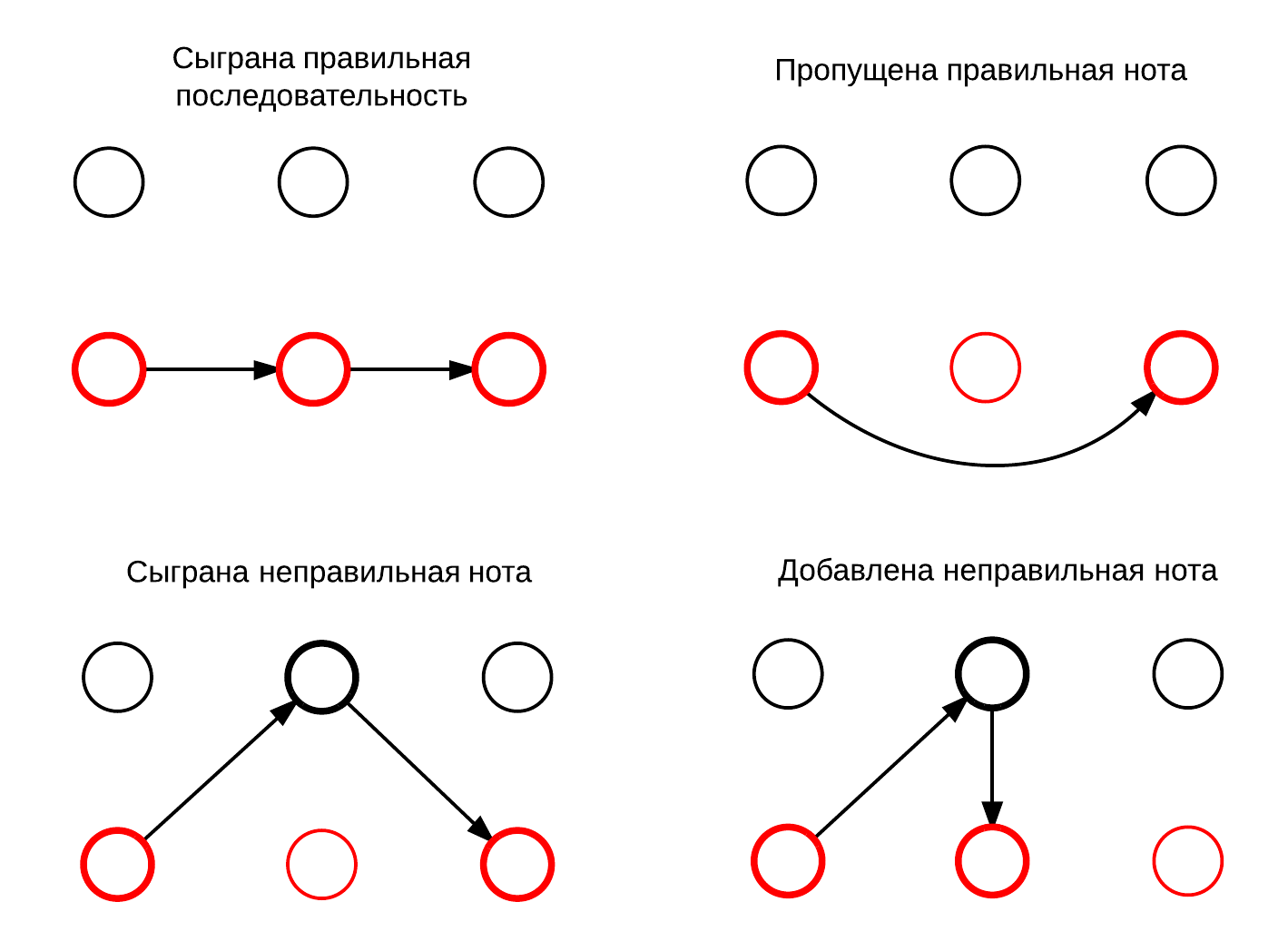


Рисунок 5 – Различные варианты ошибок исполнения.

## Обзор существующих решений синхронизации нот

### Tonara

Данный программный комплекс предназначен для синхронизации играемых нот. Эта система «слушает» музыкантов и помогает им в обучении, репетициях и исполнении различных музыкальных произведений. В ней реализованы автоматическая оценка следующих нот, определения реального темпа исполнения и адаптация под него, перелистывание нотных страниц, работа в полифоническом режиме и другие интеллектуальные функции[3]. Недостатками являются платность и закрытость технологии, а также ее непопулярность. На рисунке ниже показаны два примера работы данной системы. Данная программа может работать в полифоническом режиме. Данная система использует различные методики и алгоритмы для качественной синхронизации нот, например гибридные марковские сети, алгоритм Витерби дли поиска наиболее подходящего списка состояний, который в контексте цепей Маркова получает наиболее вероятную последовательность произошедших событий и другие. Недостатком данной системы отслеживания нот является закрытость технологии и невозможность оценки степени синхронизированности.

## Выводы по аналитическому разделу

На основе сравнения различных алгоритмов определения основного тона, было принято реализовать модифицированный алгоритм, основанный на автокорреляции, так как он не требует преобразование Фурье для входного звукового сигнала. Для синхронизации временных рядов был выбран алгоритм Витерби, работающий в Скрытых Марковских Моделях, так как такие модели хорошо исследованы в настоящее время и их можно гибко настраивать и обучать, в отличие от метода сопоставления строк. В следующем разделе будет описано как выбранные методы и алгоритмы были скомбинированы и реализованы.

# Конструкторская часть

## Модифицированный алгоритм определения ЧОТ

В итоге для определения частот был разработан эффективный алгоритм, основанный на автокорреляции, но без следующих проблем, присущих автокорреляции:

1. Автокорреляция имеет сложность что недостаточно для задач реального времени. Данная проблема была решена с помощью двух этапов:
   1. Расчет автокорреляции только для каждого восьмого замера PCM аудиоданных и определение приблизительной частоты.
   2. Просчет более точной частоты посредством более точного алгоритма высокого разрешения.
2. Автокорреляция неточная, особенно на высоких частотах.  
   Чем больше частот, тем менее аккуратно будет работать алгоритм автокорреляции. Это происходит из-за того, что форма звуковой волны на высоких частотах меньше замеров на период.  
     
   Для того, чтобы решить данную проблему второй проход в алгоритме использует интерполяцию для того, чтобы сравнить измерения, которые не ограничены целыми числами. Например, если измерение 0 будет на позиции 0.0, измерение 1 может быть на позиции 0.674 и т.д. Это позволяет шагам точно определять шаги для определения частоты в одном фрагменте, в отличие от квантовании в скачкообразные шаги. Данный алгоритм используют 4 точки для интерполяции методов Хермита.

Таким образом, был разработан следующий алгоритм для определения частоты основного тона в онлайновом и офлайновом режиме, обладающий следующими достоинствами:

1. Как было описано выше, разработанный алгоритм оказался быстрым, способным определять до 3000 частот в секунду.
2. Отклонение от реальной частоты составляло не более чем 0.02%.
3. Точность относительно широкого диапазона входных уровней громкости. Так как алгоритм использует соотношения различных пиковых значений, а не абсолютные значения, то он оказался применим для большого диапазона уровней громкости. Потерь точности нет для диапазона от -40db до 0db.
4. Точность относительно полного спектра частот. Разработанный алгоритм оказался способным распознавать частоты в диапазоне от 50 Hz до 1.6 kHz. Это оказалось благодаря интерполяции при расчете, примененной к скользящему окну.
5. Точность в отношении любого типа форм звуковых волн. Несмотря на большое количества определения частот основного тона, данный алгоритм работает одинаково для всех типов звуковых волн. Это означает что он хорошо работает с мужскими и женскими голосами, не говоря уже про музыкальные инструменты, такие как флейта, гитара и др. Единственным требованием для него является монофонический сигнал, т.е. аккорды при этом не могут распознаны. Разработанный алгоритм также может использоваться для очень чувствительного гитарного тюнера или других инструментов.
6. Разработанный алгоритм не зависит от предыдущих результатов распознавания. Каждое новое распознанная частота является новым рассчитанным значением. В некоторых других рассмотренных алгоритмах существует проблема зависимости результатов распознавания частоты от предыдущих результатов. Особенно это касается проблемы с октавами (выше или ниже).

## Разработка оптимизированной СММ.

В данном разделе описано как нотная запись может быть представлена в виде СММ. СММ – это упорядоченная пятерка, в которой N – количество возможных состояний процесса (в данном случае количество нот), M – количество возможных наблюдений при переходе от состояние в состояние (в данном случае все высоты нот или идентификаторов нот), A – матрица вероятностей перехода из состояния i в состояние j (в данном случае перехода из ноты в ноту), B – матрица вероятностей перехода из состояния i в состояние j по наблюдению o (в данном случае перехода из ноты в ноту по тону), pi – матрица вероятностей запуска состояния в процессе i (в данном случае в определенной ноте). Таким образом, одной из решенной задач было генерация матрицы **A**, **B** и **pi** на основе эмпирических величин.

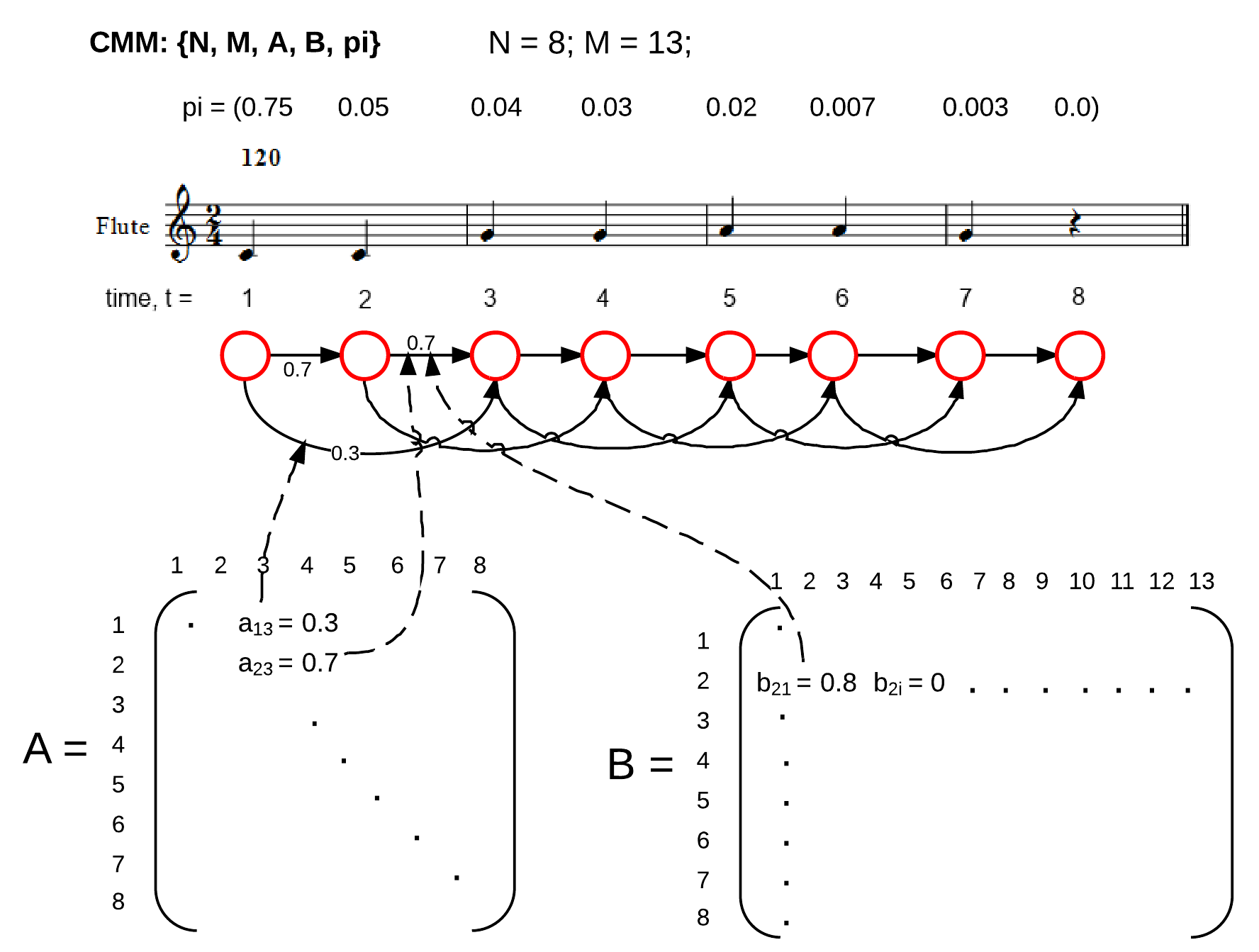


Рисунок 6 – Представление нотных записей в виде СММ.

## Равномернотемперированный музыкальный строй

При настройке инструмента разным нотам присваиваются разные частоты звучания. При этом присвоение частот можно произвести разными способами.

Существует несколько видов музыкальных строев: пифагорейский, натуральный, среднетоновый и равномерно темперированный.

**Натуральный строй** – музыкальный строй, использующий интервалы, построенные на основе обертонов (Рисунок 1.9). Октава (1:2), квинта (2:3), кварта (3:4), большая терция (4:5), малая терция (5:6), большой целый тон (8:9), малый целый тон (9:10), и диатонический полутон (15:16). В результате получается гамма абсолютно гармоничная начальному тону, но только в пределах этой тональности. И даже в пределах семиступенной гаммы есть фальшивые интервалы.

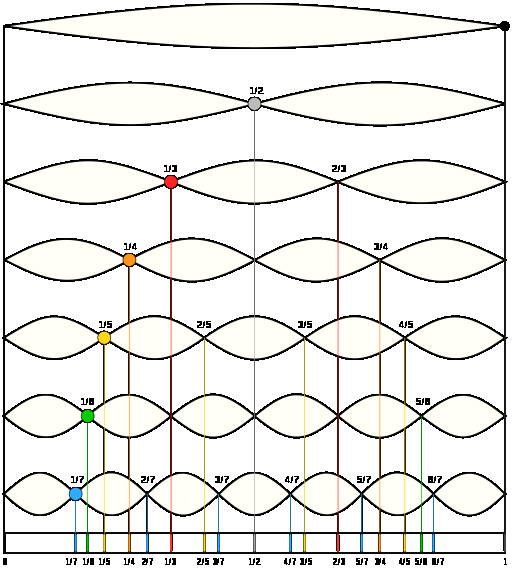


Рисунок 7 – Темперация в музыкальном строе.

**Равномерно темперированный строй** – музыкальный строй, при котором каждая октава делится на математически равные интервалы, чаще всего на двенадцать полутонов (). Такой строй господствует в европейской профессиональной музыке приблизительно с XVIII века вплоть до нашего времени[1)]. Таким образом, частоту каждой ноты можно вычислить по следующей формуле:

 (1.7)

где где *F0* — частота камертона (например Ля 440 Hz). Последовательность вычисленных таким образом частот образует геометрическую прогрессию.

По причине того, что в равномерно темперированном строе все аккорды становятся однообразными и симметричными, в то время как в старых строях из-за неравномерности темперации каждый аккорд имеет своё неповторимое (акустическое) звучание, а также в связи с тем, что сейчас равномерная темперация стала фактическим стандартом, другие строи в данном проекте рассматриваться не будут.

## Представление нот и пауз в матрице переходов

Множество возможных наблюдений представлено как диапазон чисел от 0 до 11, сопоставленные всем нотам из списка {C, C#, D, Eb, E, F, F#, G, G#, A, Bb }. Однако при моделировании более сложных нотных записей, в которых присутствуют паузы или стакатто, к этому множеству добавлялось дополнительное состояние. Преобразование в данный диапазон выполняется по формуле: .

## Алгоритм Витерби в реальном времени.

При разработки системы, задержка между входными и выходными данными имеет большое значение. Из-за большого количества состояний СММ, программа не может отслеживать позицию в реальном времени из-за большого количество времени, требуемого на вычисление алгоритма Витерби. Такой проблемы не встает если использовать наиболее последние наблюдения, которые в значительной степени влияют на следующую позицию в системе.

Для каждой новой играемой музыкантом ноты, используя 3 или более наблюдений, сложность алгоритма оценивается как O(N3). Для цепочки наблюдений из M музыкальных событий с историей X наблюдений, сложность будет оцениваться как O(M\*N2). Это неприемлемо для задачи отслеживания нот в реальном времени.

Для того чтобы сократить вычислительную нагрузку, используется таблица вероятностей, рассчитанная заранее, до того как музыкант начинает играть. Во время работы системы можно использовать недавние наблюдения в качестве ключа для извлечения наиболее вероятного следующего состояния которое должно быть.

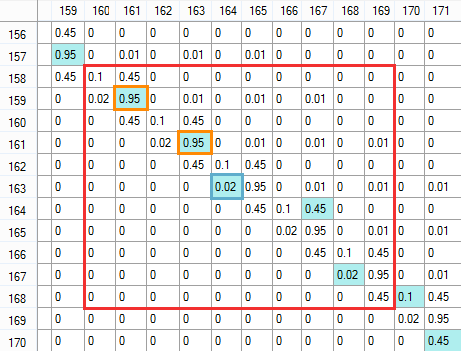


Рисунок 8 – Принцип работы алгоритма Витерби в окне.

В исследованной литературе, многие системы для отслеживания нот были разработаны для длинных участков нот (Рафаэль, 2001b; Орио, 2003; Пардо и Бирмиргам, 2005) без пересортировки.

Для того, чтобы сократить вычислительные расходы, было принято реализовать механизм скользящего окна в СММ.

Вместо того чтобы рассматривать все состояния СММ как потенциально возможные состояния, в методе будут рассматриваться локальные состояния, сконцентрированные около текущего. Данный массив состояния называется «окно». Размер окна принимается приблизительно равным размеру нескольких тактов.

Если же возникает ситуация, что система зацикливается, то размер окна увеличивается, для того, чтобы охватить больше наблюдений.

## Синхронизация музыканта и системы в начале.

Система была разработана таким образом, что процесс синхронизации не начнется до тех пор, пока не будет правильно сыгран хотя бы один квант первоначального нотного события. Также стоит отметить, что на каком-то этапе система перестает идентифицировать звуковые фрагменты как ноты (т.е. идентифицирует как паузы), то процесс синхронизации также останавливается.

## Расчет темпа в реальном времени.

Для определения степени синхронизированности нотных записей и живой музыки нужно определять не только правильное сопоставление фактических и идеальных состояний модели, но и определять аритмичность исполнения музыки. Для этого использовался алгоритм отслеживания ритма.

Отслеживания ритма – это процесс расчета темпа или скорости исполнения определенного участка нотной записи с помощью анализа прослушанной музыки. В настоящее время это активно развивающая область исследований (Диксон, 2000, MIREX, 2006а).

Было решено сделать простую реализацию алгоритма отслеживания ритма в данном проекте, по сравнению с последними достижениями. Однако получилось, что даже разработанный простой метод работает довольно эффективно.

В разработанном методе измеряется время между двумя сыгранными музыкантом нотами. Если музыкант в данный момент находится в гостевом состоянии модели (т.е. отклонился от правильной последовательности), тогда последний входной сигнал не рассматривается как правильный и не используется для обновления темпа. Если музыкант в текущий момент находится в нормальном состоянии (т.е. они могут найдены в нотах), тогда система сравнивает, сколько должна звучать предыдущая нота, а затем сравнивает это продолжительность с реально продолжительность последней ноты.

Текущий темп зависит от среднего количество измерений темпа. Наибольшие и наименьшие наблюдения за темпом при этом игнорируются.

Таким образом, были реализованы следующие задачи:

• Если музыкант находится в гостевом состоянии, это значит, что он допустил ошибку, таким образом, нота, которую он имеет в текущий момент, поэтому данная нота не рассматривается как надежная для отслеживания темпа.

• Если музыкант находится в состоянии, где он не должен играть новую ноту, тогда эта информация по определению не является значимой.

• Только входная фактическая информация, которая совпадает с идеальной, учитывается при расчете темпа. В этих случаях система может записывать длительности таких нот и сравнивать их с реальными продолжительностями (т.е. текущий тем, умноженный на количество ударов, которые содержатся в одной ноте). Система сравнивает расхождение между ожидаемой продолжительностью ноты, фактической продолжительностью и меняет собственный темп при необходимости.

Одно из самых важных решений, которое было принято реализовать в данном проекте, заключалось в представлении темпа как миллисекунды на удар, в отличие от традиционной шкалы темпа, измеряемой в ударах в минуту. Данное решение сделало вычисления более простыми и сильно не повлияло на правильность системы, так как расчет темпа главным образом используется для внутренних расчетов, а не для обратной связи с исполнителем.

Итак, шаги данного алгоритма можно описать следующим образом:

* Замер времени последних N не гостевых событий в возрастающей последовательности ()
* Измерение разницы между каждыми соседними двумя полученными событиями ()
* Удаление 2 наименьших или наибольших значений ()
* Измерение времени между полученными 6 величинами последующей формуле:

Разработанный алгоритм показал незначительно худшие результаты по сравнению с последними достижениями (Диксон, 2006).

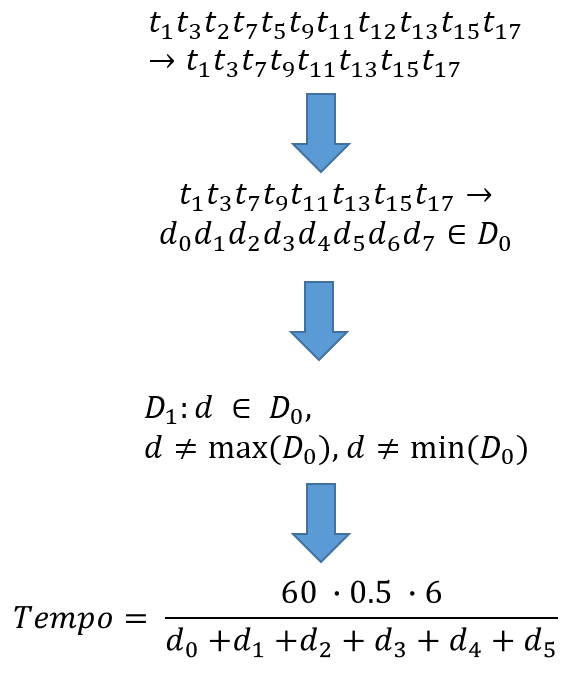


Рисунок 9 – Алгоритм определения темпа исполнения в реальном времени.

## Методика оценки степени синхронизированности.

Разработанная методика заключается в вычислении параметра, названного *штрафным баллом* который зависит от четырех агентов:

1. Позиция в нотах (для квантов).
2. Позиция в нотах (для нотных событий).
3. Текущий темп.
4. Громкость.

Схемы работы алгоритмов указанных агентов приведены ниже.

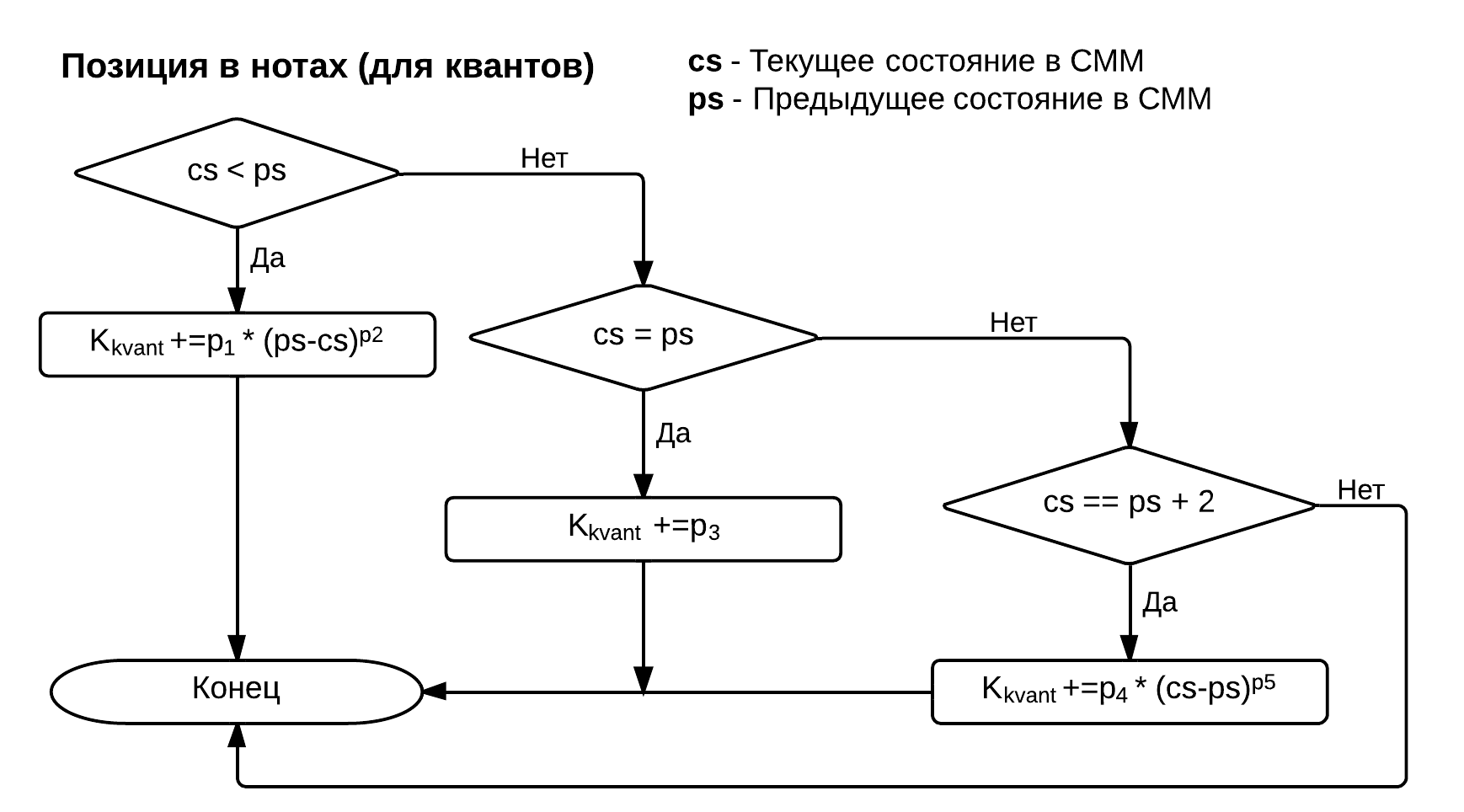


Рисунок 10 – Агент, рассчитывающий штрафной балл в зависимости от последовательности квантов.

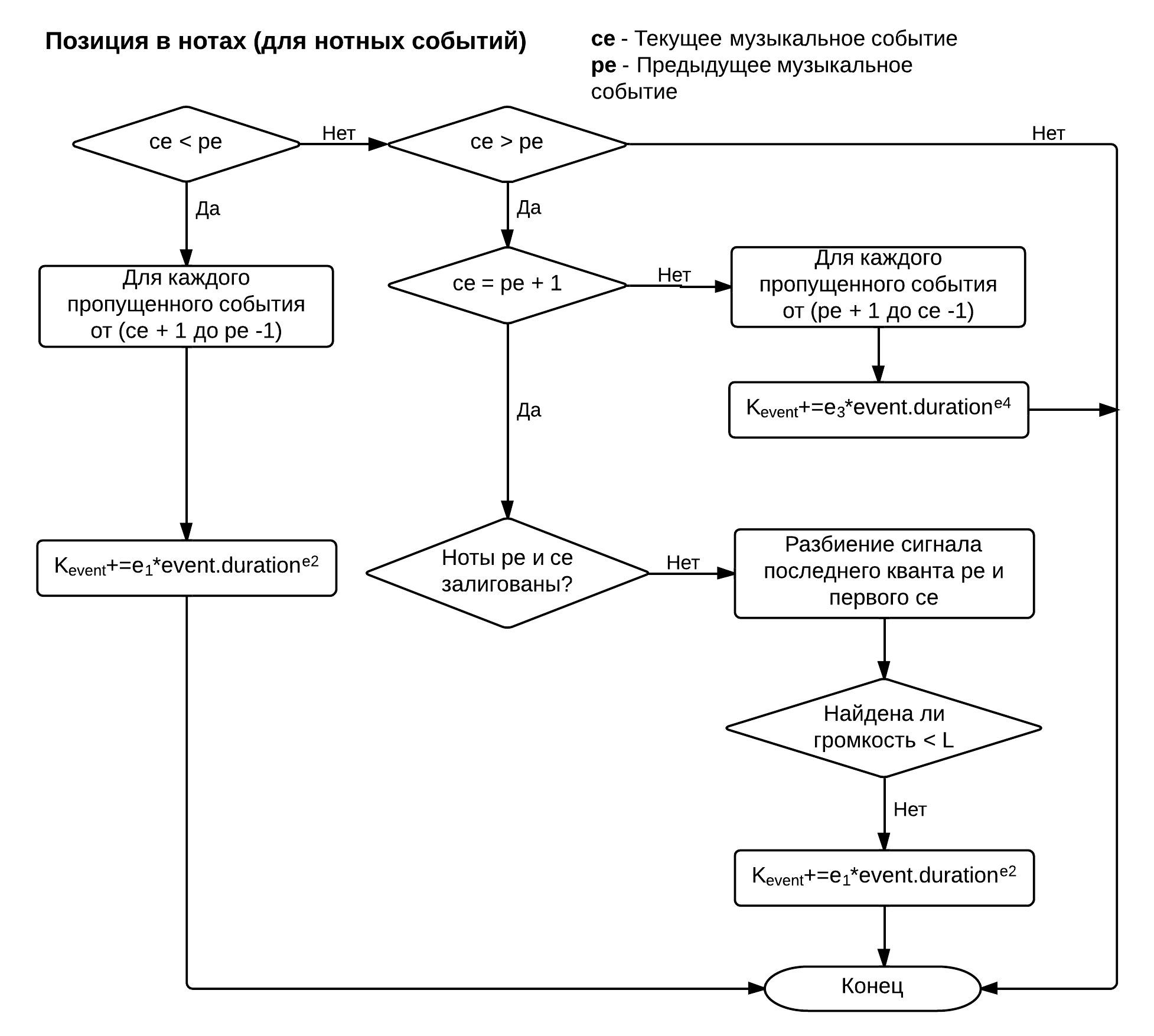


Рисунок 11 – Агент, рассчитывающий штрафные балы в зависимости от последовательности нотных событий.

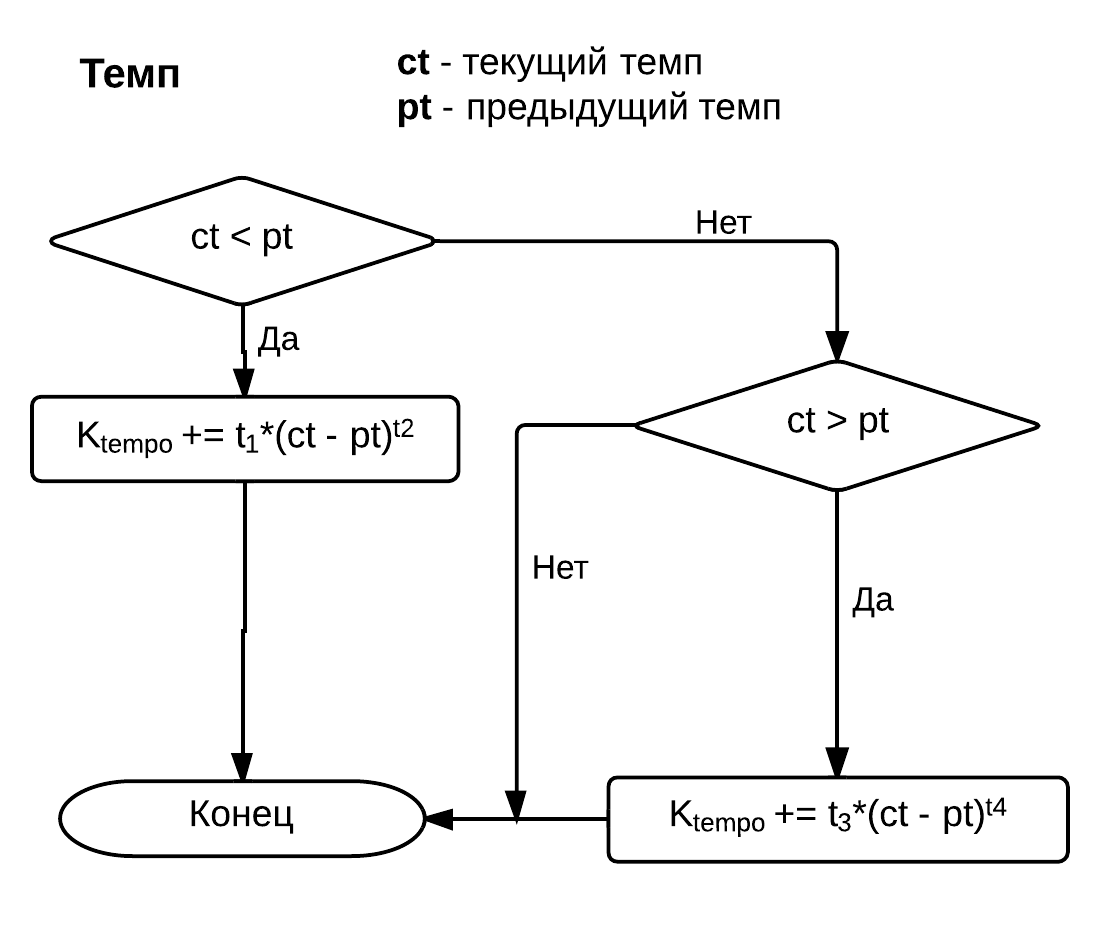


Рисунок 12 – Агент, рассчитывающий количество штрафных баллов в зависимости от изменений темпа.

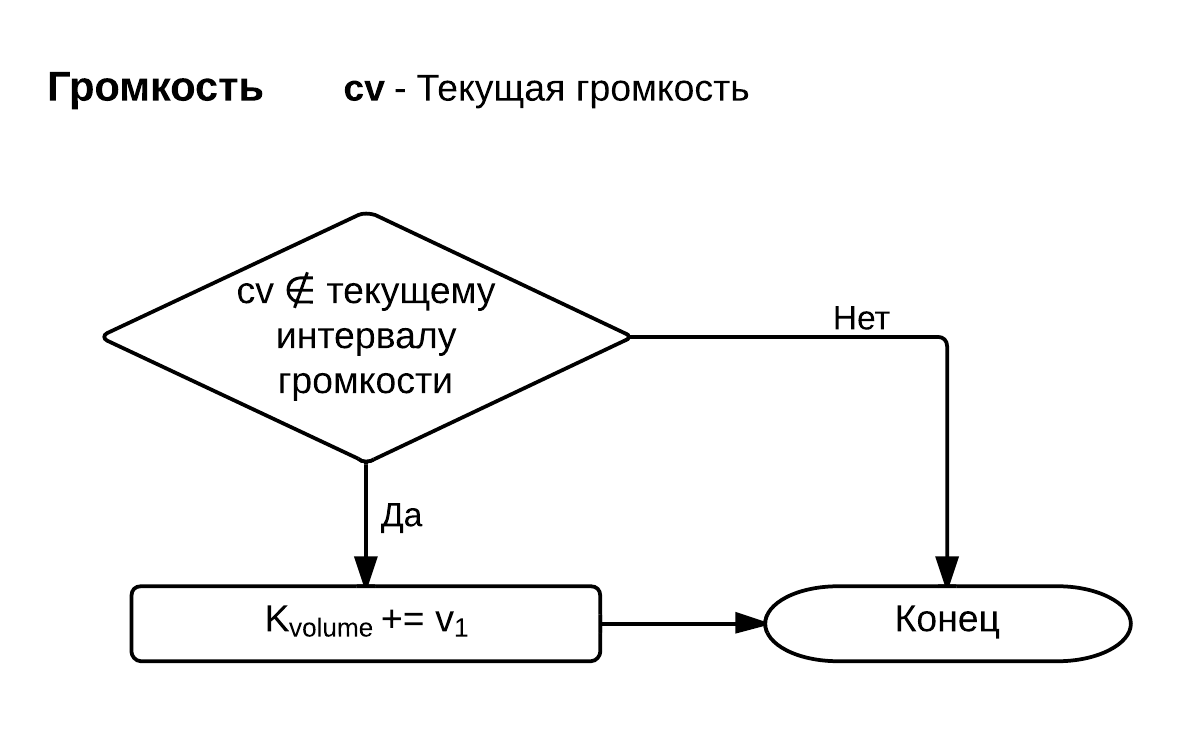
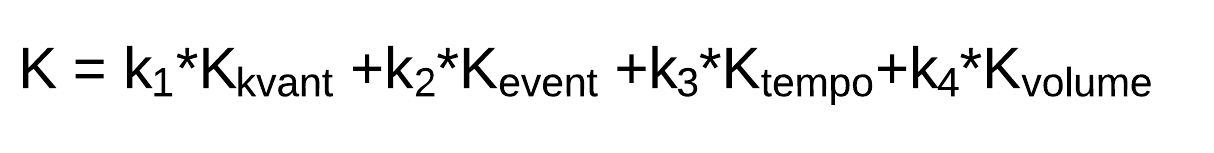


Рисунок 13 – Агент, рассчитывающий штрафные баллы в зависимости от текущей громкости.

Итоговая формула вычисления штрафных баллов:



Таким образом, разработанная методика позволяет вычислять штрафные баллы не только от правильной последовательности не гостевых состояний, но от текущего темпа исполнения и громкости.

## Выводы по конструкторскому разделу

В данном разделе были детально описаны:

* Модифицированный алгоритм вычисления частоты основного тона на основе автокорреляции.
* Скрытая Марковская Модель для представления нотных записей.
* Алгоритм Витерби в реальном времени.
* Алгоритм расчета темпа в реальном времени.
* Методика оценки степени синхронизированности в реальном времени.

В следующем разделе будет детально описан выбор средств для реализации разработанных алгоритмов и методики, а также структура разрабатываемой системы и пользовательский интерфейс.

# Технологическая часть

## Требования к вычислительной системе

Требования для приложения, выполняющего анализ музыки в онлайновом и офлайновом режимах:

* x86-совместимая архитектура;
* ОС Windows XP/Vista/7/Linux/Mac
* Не менее 1Гб оперативной памяти.
* Желательно: многоядерный процессор с частотой не менее 2ГГц.

## Выбор языка и среды программирования

Среди языков и технология для разработки серверной части были проанализированы следующие:

* Node.js (серверная реализация языка программирования JavaScript, основанная на движке V8);
* Java;
* C#.

Существуют следующие технологии для разработки клиентской части:

* Adobe Flash;
* HTML 5 + JavaScript;
* Microsoft Silverlight;
* Native Client.

Так как одной из задач работы являлась функционирование системы на как можно большем количестве устройств, то технологии Microsoft Silverlight и Native Client были сразу же отброшены, поскольку Silverlight хорошо работает только на операционных системах класса Windows, а Native Client вообще работает только в определенном браузере Google Chrome и не имеет стабильной версии.

Из Adobe Flash и HTML 5 + JavaScript выбор был остановлен на последней, поскольку Flash, несмотря на большую распространенность, не работает на операционных системах iOS. Кроме того, не существует оптимального инструмента для универсальной разработки серверных и клиентских частей для Flash.

Проанализировав различные подходы к разработке серверной и клиентской частей веб-приложений, был выбран язык программирования C# на сервере, при этом компилирующийся в JavaScript на клиенте. Данный выбор обуславливается более совершенным синтаксисом, возможностью компиляции C# не только под JavaScript, но и под такие платформы как Windows Phone, Android и iOS с помощью , что потенциально может понадобиться в будущем, а также возможностью использования вставок неуправляемого кода для достижения наибольшей производительности.

## Трансляция C# в JavaScript

Существует несколько способов трансляции языка C# или сборок .NET в код JavaScript:

* JSIL
* SharpKit
* Script#
* Другие

Script# оказался оптимальным выбором, поскольку JSIL работает непосредственно со сборками и генерирует менее чистый код, хотя и поддерживает больше возможностей C#, а SharpKit является коммерческим. Подробное сравнение подобных инструментов описано в (4).

Резюмируя информацию из предыдущих двух разделов, можно выделить следующие достоинства и недостатки использования технологий C# и Script# для разработки универсального кода под серверную и клиентские части разрабатываемого веб-приложения:

**Достоинства:**

• Возможность написания универсального кода под .NET и другие платформы (WP, Mono).

• Разработка на строго типизированном языке C# с возможностями ООП.

• Поддержка возможностей IDE для разработки (автодополнение, рефакторинг).

• Определение многих ошибок на этапе компиляции.

**Недостатки:**

• Избыточность и нестандартность генерируемого JavaScript (из-за mscorlib).

• Поддержка только спецификации ISO-2 (отсутствие перегрузки функций, вывода типов, расширений, обобщений и другого).

Итак, схема компиляции под различные целевые платформы представлена на рисунке ниже.

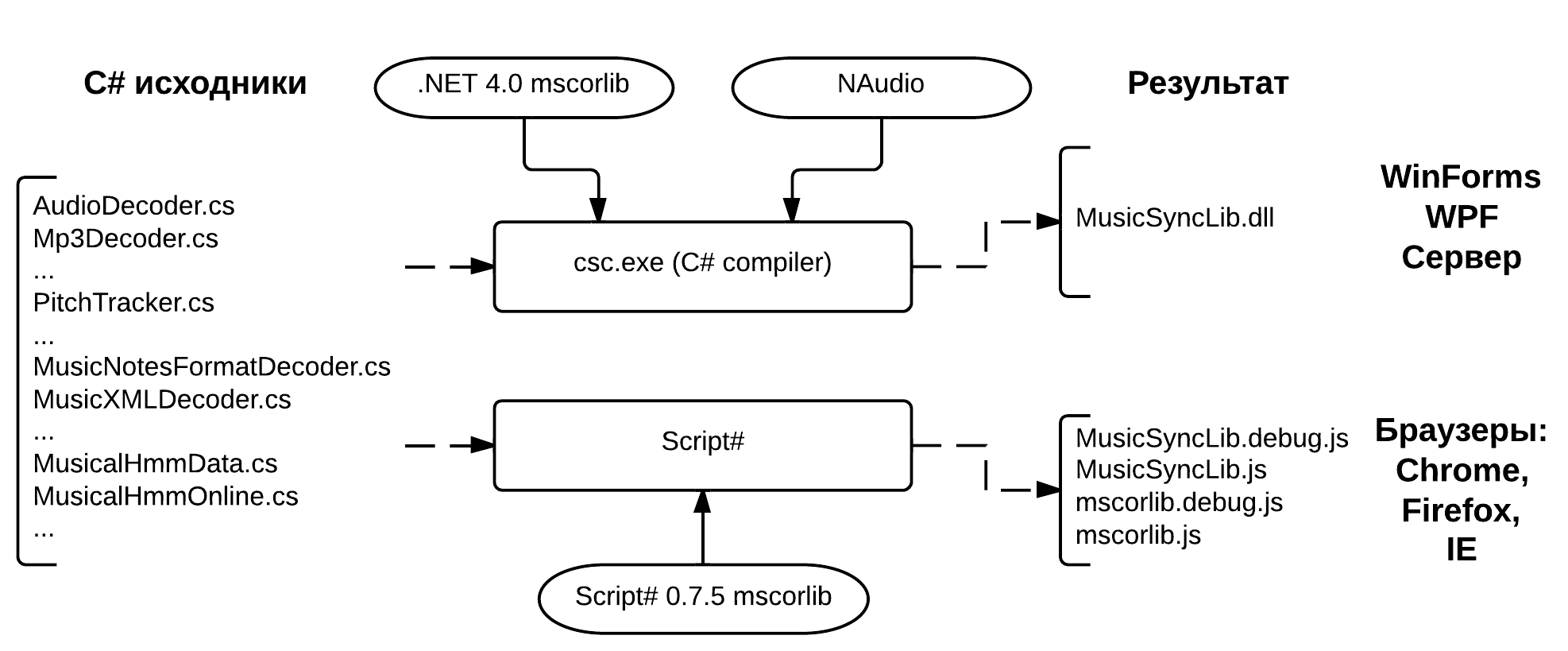


Рисунок 14 – Схема компиляции кода C# под различные целевые платформы.

## Структура разрабатываемой системы

При разработке программного продукта применялся объектно-ориентированный подход. В данном разделе будут описаны основные классы приложения и представлены взаимосвязи между ними.

Диаграмма классов, содержащих основную логику создания нотаций в реальном времени.

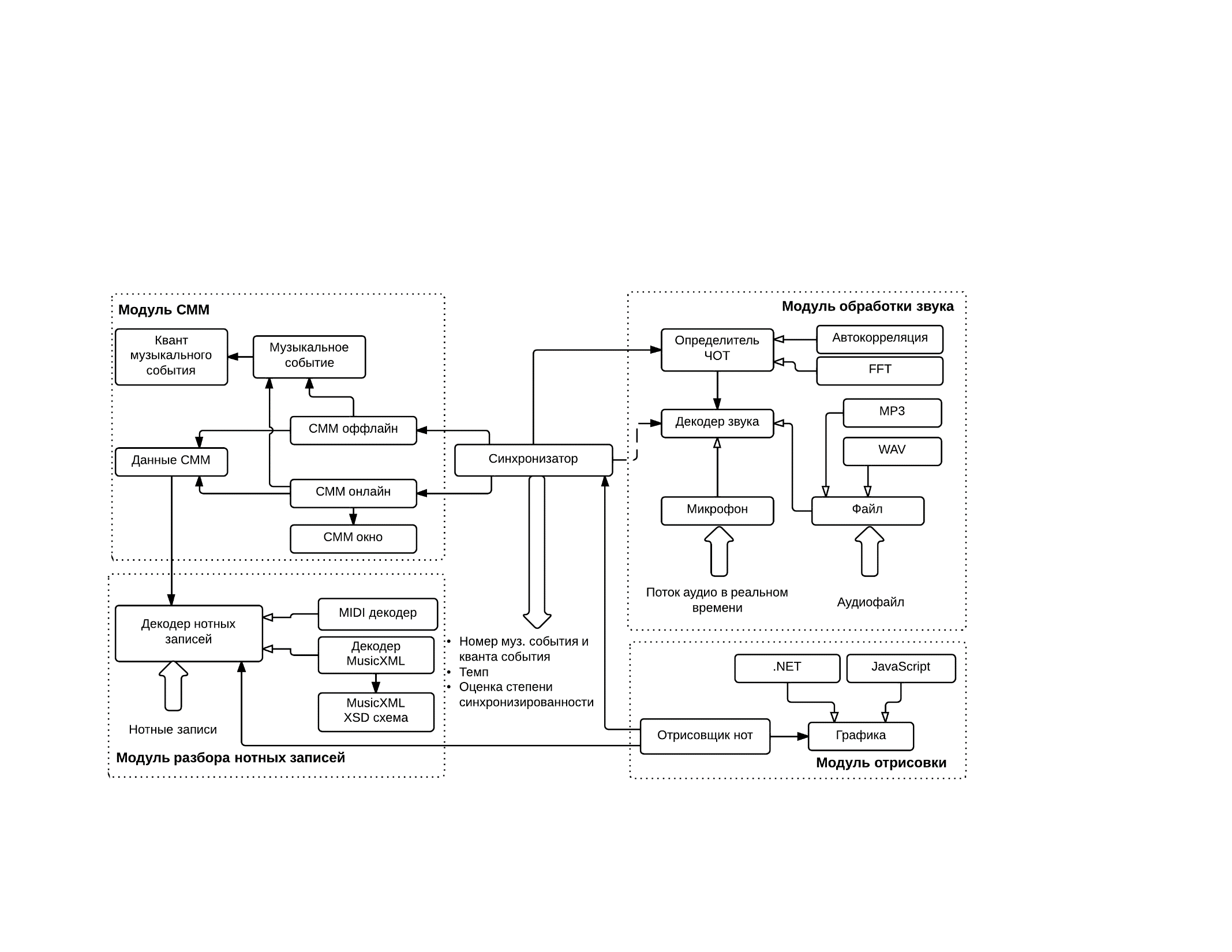


Рисунок 15 – Структура разрабатываемой системы.

## Модуль определения частоты основного тона

**PitchTracker** – Определяет частоту основного тона. Он используется следующим образом.

PitchTracker pitchTracker = new PitchTracker();

pitchTracker.SampleRate = 44100.0;

pitchTracker.ProcessBuffer(audioBuffer);

Таким образом, существует три способа возврата частоты из данного класса:

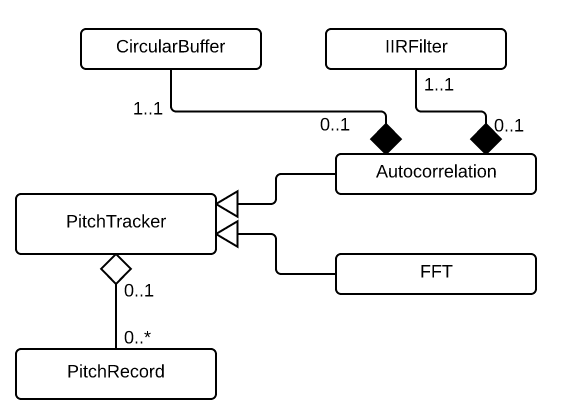
1. Использование **PitchTracker.CurrentPitchRecord для определения последней определенной частоты.**
2. **Использование буфера истории, посредством установки флага PitchTracker.RecordPitchRecords**. Данный флаг позволяет добавлять распознанные частоты во внутренний буфер. При этом буфер можно лимитировать с помощью свойства **PitchTracker.PitchRecordHistorySize. Это создаст скользящий буфер, который например удобно отображать на диаграмме. Стоит отметить, что если частота не была распознана (например, пауза), то запись в буфер все равно добавляется.**
3. **Подписка на событие PitchTracker.PitchChanged. Данное событие будет вызываться каждый раз, когда новая частота будет распознана. В данном случае тоже если частота все же не была распознана, то данное событие все равно вызывается.**

Не существует никаких ограничений, касающихся размера буфера, содержащего распознанные частоты.

**CircularBuffer –** кольцевой буфер. Эта структура данных, использующая единственный буфер фиксированного размера, как будто бы после последнего элемента сразу же снова идет первый. Такая структура легко предоставляет возможность буферизации потоков данных.

**IIRFilter –** фильтр с бесконечной импульсной характеристикой. Линейный электронный фильтр, использующий один или более своих выходов в качестве входа, то есть образует обратную связь. Основным свойством таких фильтров является то, что их импульсная переходная характеристика имеет бесконечную длину во временной области, а передаточная функция имеет дробно-рациональный вид. Используется в классе **PitchTracker** для определения частоты основного тона.

**PitchRecord –** структура, содержащая запись об определенной частоте. Используется в **PitchTracker.**



## Модуль для обработки MusicXML файлов

**MusicXmlScheme** – автоматически сгенерированная xsd схема для формата MusicXML.

**MusicXmlNotesExtractor –** класс, извлекающий (квантующий) последовательность музыкальных событий из **MusicXmlScheme.**

**MusicalXmlNode –** структура, содержащая фрагмент данных в формате MusicXml.

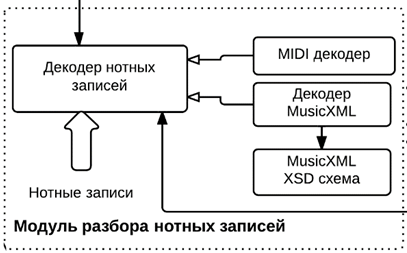


Рисунок 16 – Структура модуля разбора нотных записей.

## Модуль для отрисовки нот

**Graphics** – абстрактный класс, содержащий в себе методы для отрисовки графических примитивов. Содержит следующие методы:

**GraphicsNET –** реализация класса Graphics для отрисовки графических примитивов под платформу .NET.

**GraphicsJS –** реализация класса Graphics для отрисовки графических примитивов под платформу JavaScript.

**MusicalNotesRender –** абстрактный класс, содержащий в себе методы для отрисовки музыкальных нотаций.

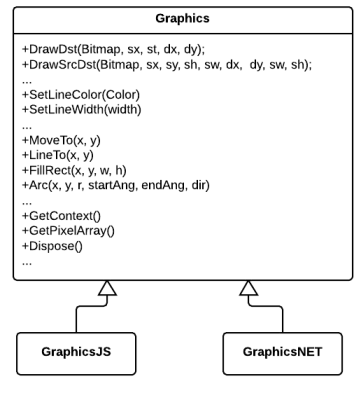


Рисунок 17 – Абстрагирование класса отрисовки графики для различных целевых систем.

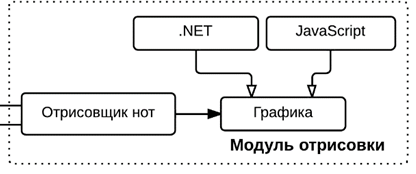


Рисунок 18 – Связь модуля отрисовки с остальной системой.

## Модуль для обработки аудио в офлайн и онлайн режимах

**Decoder –** класс, использующийся для декодирования аудиофайлов в формате **.wav** или **.mp3** для извлечения звуковых фрагментов.

**DecoderRealTime –** класс, использующийся для захвата аудиопотока в реальном времени.

**Sample –** класс, содержащий звуковой фрагмент из аудиопотока в офлайн или онлайн режимах в формате PCM.



Рисунок 19 – Схема модуля обработки звука.

## Модуль для анализа нотных записей.

**MusicalHMMData –** класс, генерирующий матрицы **A**, **B** и **pi** для использования их в алгоритме Витерби в обычном времени и нет.

**MusicalHMMOffline –** класс для синхронизации звуковых фрагментов, поступающих из модуля для обработки аудио и модуля обработки MusicXml данных в обычном времени.

**MusicalHMMOnline –** класс для синхронизации звуковых фрагментов, поступающих из модуля для обработки аудио и модуля обработки MusicXml данных в реальном времени.

**MusicalHMMWindow** – класс, рассчитывающий окно состояний, для функционирования алгоритма Витерби в реальном режиме.

**MusicalEvent –** структура, содержащая информацию о музыкальном событии.

**MusicalKvantEvent –** структура, содержащая информацию о кванте музыкального события.

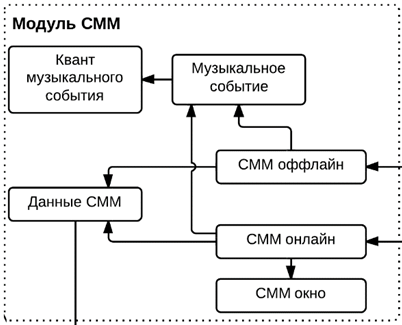


Рисунок 20 – Структура модуля СММ.

## Пользовательский интерфейс

Пользовательский интерфейс реализован посредством технологии WinForms.

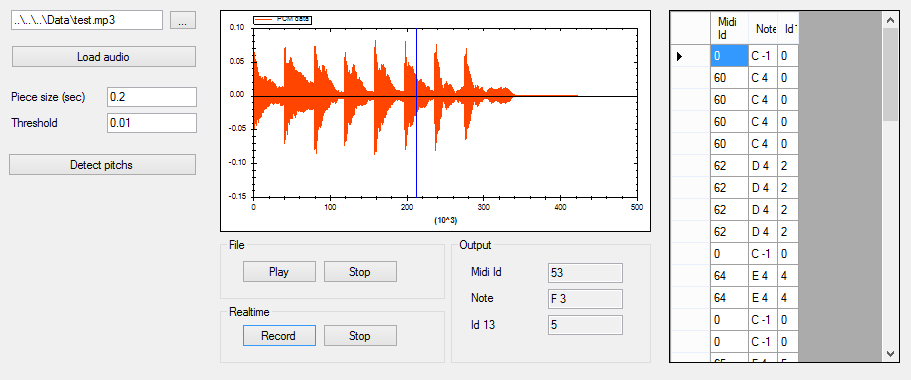


Рисунок 21 – Форма для тестирования алгоритмов определения частоты основного тона.

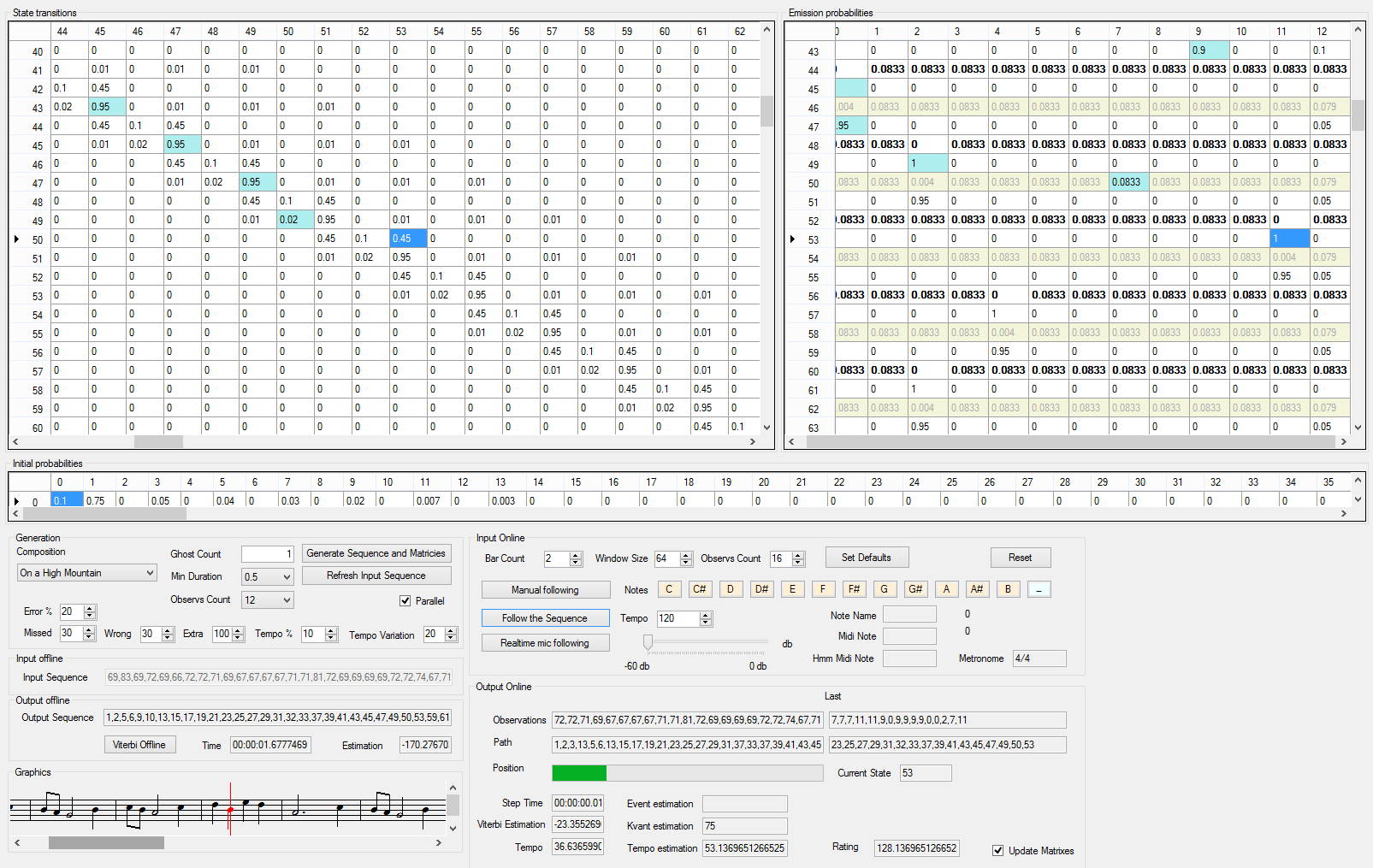


Рисунок 22 – Главное окно приложения.

На снимке экрана выше представлена программа в момент выполнения:

Слева вверху отображается матрица вероятностей перехода из состояния в состояние. Справа вверху отображается матрица вероятностей перехода из состояния в состояние по определенному наблюдению. Внизу слева представлены настройки для генерации последовательности квантов для тестирования с ошибками и без. Внизу справа отображается текущий процесс синхронизации в реальном времени. Предусмотрено тестирование на основе сгенерированной последовательности, а также на основе входного аудиосигнала. Также внизу отображается оценка степени синхронизированности.

## Описание форматов файлов

### Midi

**MIDI** – это стандарт цифровой звукозаписи на формат обмена данными между электронными музыкальными инструментами. Для того, чтобы иметь возможность прослушивать сформированную нотную запись, необходимо сохранить ее в формате MIDI, который поддерживается всеми современными аудио устройствами и программами.

### MusicXML

**MusicXML** – открытый формат файлов музыкальных нотаций, основанный на формате XML. Он разработан специально для взаимодействия разных нотных редакторов между собой. В мае 2013, MusicXML поддерживало более 140 нотных программ, включая большинство нотных редакторов, программ распознавания нот, секвенсоров и других. В приложении Г приведен пример музыкальной партии, записанной в этом формате.

## Используемые сторонние библиотеки

### NAudio

**NAudio** – библиотека с открытым исходным кодом под .NET для обработки аудио: преобразования аудиофайлов в формат pcm, захвата аудиопотока с микрофона и т.д.

# Экспериментально-исследовательский раздел

В данном разделе описаны проведенные эксперименты, призванные определить точность разработанной системы при синхронизации нотных записей, а также определить необходимые для работы системы параметры.

## Тестирование алгоритмов определения ЧОТ

В данном эксперименте были проанализированы алгоритмы определения частоты основного тона, рассмотренные в аналитическом разделе, такие как:

* Алгоритм пересечения с нулем
* Алгоритм автокорреляции.
* Гармоническое перемножение спектров.
* Модифицированный алгоритм автокорреляции.

Первый эксперимент заключался в определении абсолютной ошибки в зависимости от частоты.

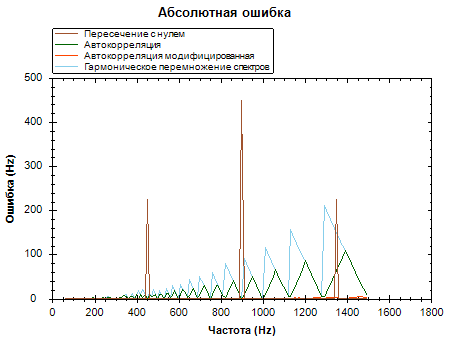


Рисунок 23 – зависимость абсолютной ошибки от частоты.

Из рисунка можно сделать вывод, что абсолютная ошибка определения частоты основного тона практически не зависит от количества отсчетов для разработанного и реализованного модифицированного метода автокорреляции.

В следующем эксперименте вышеуказанные алгоритмы тестировались с точки зрения времени вычисления.

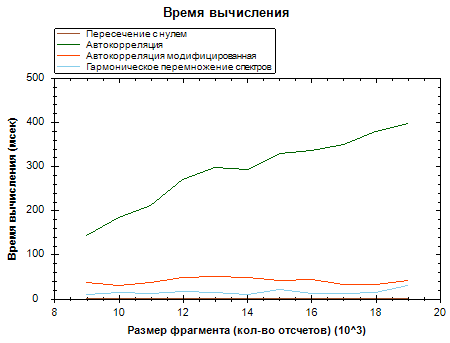


Рисунок 24 – Зависимость времени вычисления от размера фрагмента.

Как можно заметить из рисунка выше, все алгоритмы показали константную вычислительную сложность в зависимости от количества отсчетов аудиофрагмента, за исключением автокорреляции. Как и ожидалось, алгоритм пересечения с нулем показал наименьшую задержку, это объясняется небольшим количеством вычислений.

## Тестирование алгоритма Витерби в реальном времени.

Для тестирования алгоритма Витерби в реальном времени провелось два теста. Первый заключался в измерении задержки обработки в зависимости от количества квантов для композиции «On a High Mountain» в темпе 120 ударов в минуту.

Рисунок 25 – Время обработки в зависимости от количества квантов.

Тестирование показало, что оптимальное количество квантов, после которого задержка начинает ощущаться равняется 32. С другой стороны, чем меньшее количество квантов обрабатывается, тем менее точным будет процесс синхронизации. При небольшом количестве система вообще может зависнуть в одном состоянии или наоборот трактовать всю поступающую входную информацию как правильную.

Рисунок 26 – Ошибка алгоритма Витерби в реальном времени (по сравнению с обычным).

Из данного теста можно заключить, что при увеличении количества квантов, принимаемых в расчет, ошибка алгоритма Витерби для нотных записей снижается. Таким образом можно заключить, что разработанный алгоритм применим для рассматриваемой предметной области.

## Тестирование методики оценки степени синхронизированности.

Для тестирования использовался метод, используемый на конференции MIREX по исследованию музыки. Данный метод был разработан с учетом нескольких важных соображений. Также возможно произвести сравнение между разрабатываемой системой и другими системами отслеживания нот.

Для тестирования метода были исследованы следующие характеристики:

* **Количество событий**. Количество музыкальных событий, включенных в исполняемую мелодию.
* **Количество пропущенных нот**. Ноты, которая система не отследила вовсе или которые были распознаны с задержкой более чем 2000 миллисекунд.
* **Ложноотрицательные срабатывания**. Ноты, которая система распознала с задержкой более чем 2000 миллисекунд (это также включено в **Количество пропущенных нот**).
* **Среднее смещение**. Среднее значение всех измерений смещения между позициями нот и правильными нотами.
* **Стандартное отклонение смещения**. Стандартное отклонение вышеописанного смещения.
* **Средняя задержка**. Среднее значение записанной задержки между двумя нотами, сыгранных исполнителем и нотами, обработанной системой.
* **Пропущенные ноты %**. Процент пропущенных нот (Numberof NotesMissed / EventCount). Отображает насколько точно система отслеживает точную позицию музыканта.
* **Ложноотрицательные срабатывания %.** Процент ложноотрицательно определенных нот. (NumberOfFalsePositives / EventCount).
* **Точность для всех композиций.** Процент корректно определенных нот в целом.
* **Точность для определенной композиции в разных тестах**. Средний процент корректно определенных нот для определенной композиции.

Также были использованы следующие критерии для синхронизации:

* 5 – Безупречное исполнение.
* 4 – Очень хорошее исполнение, практически без ошибок.
* 3 – Хорошее исполнение с несколькими ошибками, но в целом точное и музыкальное.
* 2 – Некоторые участки исполнены с большим количеством ошибок или немузыкально.
* 1 – Слабое исполнение, только в нескольких моментах сыграно точно.
* 0 – Исполнение совсем не похоже на то, что должно быть.

Все характеристики описанные выше использовались в объективном тестировании с помощью разрабатываемой системы.

Были протестированы следующие мелодии:

1. Twinkle Twinkle Little Star. Включено отслеживание ритма.
2. All I Ask of You. Включен метроном.
3. All I Ask of You. Включено отслеживание ритма. Таким образом, система должна следовать за темпом.
4. Danse Macabre. Включено отслеживание ритма. Используются вероятности, рассчитанные с помощью алгоритма Витерби в офлайновом режиме.
5. Danse Macabre. Включено отслеживание ритма. Используются вероятности, рассчитанные с помощью алгоритма Витерби в онлайновом режиме. Используется история из последних трех наблюдений за музыкантом.
6. Danse Macabre. Включено отслеживание ритма. Используются вероятности, рассчитанные с помощью алгоритма Витерби в онлайновом режиме. Используется история из последних четырех наблюдений за музыкантом.

Пять тестов было проведено на разных системах отслеживания нот. Для каждого теста была представлена определенная мелодия от музыканта. В течение каждого теста, исполнение было оценено по критериям, описанным выше. Были проведены следующие пять тестов:

1. Исполнение мелодии без ошибок, изменений темпа или украшений.
2. Исполнение мелодии с определенными ошибками.
3. Исполнение мелодии с определенными украшениями.
4. Исполнение мелодии с определенными изменениями темпа.
5. Исполнение мелодии со всем изменениями, указанными в пунктах 2, 3 и 4.

|  |  |  |
| --- | --- | --- |
| Системы отслеживания нот | Точность для всех композиций | Средняя точность для одной конкретной композиции |
| Ариша Конт и Диемо Шварц (MIREX 2006) | 82.90% | 90.06% |
| Миллер Пукетт (MIREX 2006) | 29.75% | 69.74 % |
| Разрабатываемая система | 60.89% | 54.04% |

Необходимо было добавить несколько согласований в разрабатываемую систему, чтобы оценочная информация могла быть собрана. Для этих согласований не пришлось менять программу, кроме добавления функции, собирающей информацию на различных этапах выполнения программы.

Входная информация поступала в автоматическом режиме.

## Результаты

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Композиция | Кол-во событий | Кол-во пропущенных нот | Ложноотр. | Сред. Смещ. | Среднеквадр. Смещ. | Средняя Задержка | % пропущенных нот | Ложноотр. % | Рейтинг |
| 1 | Twinkle | 7 | 0 | 0 | 20,95 | 3,99 | 0.07 | 0.00% | 0.00% | *5* |
| 2 | Twinkle | 7 | 0 | 0 | 12,81 | 8,04 | 0,08 | 0.00% | 0.00% | *5* |
| 3 | Twinkle | 7 | 1 | 0 | 52,08 | 70.91 | 0,08 | 14.29% | 0.00% | *3* |
| 4 | Twinkle | 7 | 0 | 0 | 95,03 | 149.54 | 0.11 | 0.00% | 0.00% | *5* |
| 5 | Twinkle | 7 | 0 | 0 | 131,14 | 180.75 | 0.07 | 0.00% | 0.00% | *3* |
| 1 | All I ask v.l | 40 | 1 | 0 | 532,15 | 1028.59 | 0,07 | 2.50% | 0.00% | *5* |
| 2 | All I ask v.l | 44 | 24 | 11 | 487,00 | 984.42 | 0,09 | 54.55% | 25.00% | *4* |
| 3 | All I ask v.l | 40 | 28 | 26 | 504.97 | 1033.88 | 0,06 | 70.00% | 65.00% | *3* |
| 4 | All I ask v.l | n/a |  |  |  |  |  |  |  |  |
| 5 | All I ask v.l | 44 | 38 | 27 | 504.47 | 1034.04 | 0.47 | 86.36% | 61.36% | *3* |
| 1 | All I ask v. 2 | 40 | 3 | 0 | 505.01 | 490.95 | 247.76 | 7.50% | 0.00%j | *3* |
| 2 | All I ask v. 2 | 44 | 11 | 2 | 433.99 | 354,00 | 239,58 | 25.00% | 4.55% | *2* |
| 3 | All I ask v. 2 | 40 | 6 | 0 | 790.79 | 555.08 | 212,33 | 15.00% | 0.00% | *2* |
| 4 | All I ask v. 2 | 40 | 13 | 1 | 473.62 | 615.65 | 158,30 | 32.50% | 2.50% | *3* |
| 5 | All I ask v. 2 | 44 | 16 | 2 | 840.37 | 547.21 | 237.38 | 36.36% | 4.55% | *2* |
| 1 | All I ask v.3 | 40 | 1 | 0 | 542.82 | 896.06 | 0,18 | 2.50% | 0.00% | *4* |
| 2 | All I ask v.3 | 44 | 23 | 11 | 499.74 | 860.74 | 0.16 | 52.27% | 25.00% | *3* |
| 3 | All I ask v.3 | 40 | 18 | 2 | 354.12 | 691.49 | 0.75 | 45.00% | 5.00% | *2* |
| 4 | All I ask v.3 | n/a |  |  |  |  |  |  |  |  |
| 5 | All I ask v.3 | 44 | 33 | 13 | 371.42 | 681.79 | 0.86 | 75.00% | 29.55% | *2* |
| 1 | All I ask v.4 | 40 | 3 | 0 | 452,11 | 458.27 | 221,50 | 7.50% | 0.00% | *2* |
| 2 | All I ask v.4 | 44 | IS | 4 | 374.66 | 312.95 | 224.47 | 34.09% | 9.09% | *2* |
| 3 | All I ask v.4 | 40 | 10 | 0 | 417.98 | 365.59 | 303,54 | 25.00% | 0.00% | *2* |
| 4 | All I ask v.4 | 40 | IS | 2 | 379.09 | 443.77 | 184.66 | 37.50% | 5.00% | *3* |
| 5 | All I ask v.4 | 44 | 20 | 4 | 358.36 | 392.61 | 215,13 | 45.45% | 9.09% | *2* |
| 1 | Danse Macabre v. 1 | 100 | 95 | 11 | 269.65 | 518.02 | 204,53 | 95.00% | 11.00% | *2* |
| 2 | Danse Macabre v.l | 113 | 108 | 15 | 283.17 | 524.4 | 301,38 | 95.58% | 13.27% | *2* |
| 3 | Danse Macabre v.l | 100 | 95 | 11 | 319.76 | 539.61 | 209.86 | 95.00% | 11.00% | *2* |
| 4 | Danse Macabre v.l | 100 | 95 | 13 | 271,83 | 547.69 | 182.42 | 95.00% | 13.00% | *3* |
| S | Danse Macabre v. 1 | 113 | 108 | 13 | 263.92 | 552.29 | 265.39 | 95.58% | 11.50% | *1* |
| 1 | Danse Macabre v.2 | 100 | 53 | 4 | 242.62 | 53.04 | 86.97 | 53.00% | 4.00% | *2* |
| 2 | Danse Macabre v.2 | 113 | 98 | 3 | 250.86 | 81.96 | 98.63 | 86.73% | 2.65% | *1* |
| 3 | Danse Macabre v.2 | 100 | 40 | 6 | 267.67 | 95.34 | 102.2 | 40.00% | 6.00% | *2* |
| 4 | Danse Macabre v.2 | 100 | 64 | 4 | 240.06 | 62.09 | 81.34 | 64.00% | 4.00% | *3* |
| 5 | Danse Macabre v.2 | 113 | 77 | 3 | 254,45 | 44.72 | 153.47 | 68.14% | 2.65% | *1* |
| 1 | Danse Macabre v.3 | 100 | 4 | 0 | 912,41 | 283.54 | 62,40 | 4.00% | 0.00% | *3* |
| 2 | Danse Macabre v.3 | 113 | *34* | 1 | 961.57 | 369.91 | 58.79 | 34.51% | 0.88% | *2* |
| 3 | Danse Macabre v.3 | 100 | 87 | 5 | 962.86 | 366.23 | 76.67 | 87.00% | 5.00% | *0* |
| 4 | Danse Macabre v.3 | 100 | 79 | 17 | 969.97 | 312.31 | 95.08 | 79.00% | 17.00% | *2* |
| 5 | Danse Macabre v.3 | 113 | 10S | 0 | 982.47 | 386.88 | 101,94 | 92.92% | 0.00% | *0* |

Таблица 1 – результаты экспериментов

## Анализ результатов тестирования

Несмотря на различные временные сроки разработки, разработанную систему можно сравнивать с системами, представленными на MIREX 2006. Слабейший результат на точность для конкретной композиции был на Danse Macabre.

Эти сравнения, тем не менее, могут быть на очень общем уровне. Сложно сравнить эти системы, так как они тестировались на различных композициях.

MIREX 2006 набор тестов содержал отрывок на флейте, сонату на скрипке, кларнет Моцарта с концертом. Эти композиции были гораздо большей длительности, чем протестированные в разработанной системе, содержащие в среднем 2239 событий (в разработанной системе в среднем 49). Стоит отметить что степень несоотвествия в синхронизации нот оказалась такой же, как и на тестовых примерах в MIREX. Таким образом, аккуратность следования зависит от того, какая композиция исполняется в конкретный момент.

Колонка **«Средняя задержка»** отображает насколько долго система система принимает входной сигнал и обрабатывает его для использования в СММ. Колонка **«Среднее смещение»** отображает время оценки наиболее вероятного состояния.

Запись числе **ложноположительных нот** вместе с **общим количеством пропущенных нот** для конкретной мелодии оказалась полезной; высокий процент ложноотрицательных нот относительно пропущенных нот в целом отображает ситуацию, где система не определила точную позицию музыканта в нотах, но где следование за музыкантом осуществляется грубо.

Как и ожидалось, в ситуациях, где включен метроном, задержка, связанная с приемом и обработкой игры музыканта была больше, чем для простого нотного следования.

В общем, в композиции, у которых оказались наименьшие значения в колонках **Missed Note %** и **False Positive %**, рейтинг был наибольшим. Это очевидный факт: в случае, если система находит корректную позицию музыканта, рейтинг не уменьшается.

Менее очевидное дополнение заключается в том, что если фиксируется большое количество пропущенных нот, но процент нот, идентифицированных после каждой задержки (ложноотрицательные ноты) больше, чем средний, тогда система оценивается как синхронизирующей хорошо. Например, в тесте 2 первой версии *All I Ask of You,* рейтинг оценился как 4. Это не смотря на более чем 50% нот, играемых музыкантом, не были сопоставлены с загруженными нотами. В отличие, например, от теста 1 для второй версии *Danse Macabre,* в которой система плохо отслеживала позицию музыканта, но рейтинг был оценен как 2. 25% нот было идентифицированы после определенной задержки (ложноотрицательные). Это означает что в тесте 2 для композиции *All I Ask of You,* система не могла точно определить позицию музыканта.

Как и ожидалось, на более простых мелодиях следование получилось значительно точным, нежели на более сложных и длинных мелодиях. В частности, на композициях *Twinkle Twinkle Little* и *Star and All I Ask of You* рейтинг был больше, чем ожидалось в тесте 5. Это был тест, в котором все ошибки из предыдущих тестов были скомбинированы в одном месте. Для этого теста использовалось много тактов, унаследованных от мелодии практически неотличимой от оригинально.

Некоторые результаты для *Danse Macabre* оказались особенно слабыми. Плохие показания получились из-за того, что система рассматривала только следующие возможные состояния, а не локальные.

В целом, если бы количество состояний было меньшим, то система, вероятно, лучше бы определяла позицию музыканта. Чем меньше состояний, тем более вероятно определение корректного следующего состояния.

Вторая и третья версии *Danse Macabre* были основаны на игре в линейном стиле без больших прыжков. Тесты 2 и 5 включают в себя прыжки на два такта назад и один вперед. Оба эти теста были обработаны хорошо, несмотря на то, что в данном случае легче потерять позицию. В третьей версии система хорошо справилась с релокацией в тесте 2.

В композиции *Danse Macabre* система сработала аккуратно в первом тесте (где соло-мелодия была исполнена корректно). И была довольно аккуратна во втором тесте. Это показывает, что чем больше информации поступает от музыканта, тем большей точности можно достичь.

В общем и целом, чем больше значения среднего смещения и задержки, тем менее музыкальным и аккуратным будет результат синхронизации нот.

В тестах наибольшими числами для среднего смещения оказались для композиции *Danse Macabre 3.* Это отразилось на рейтинга, особенно в тестах 3 и 5. Тем не менее, общая аккуратность для некоторых таких тестов оказалась значительно больше, чем ожидалось. Система находила следующие состояния. Вероятно, без объективного тестирования это не было бы замечено.

# Заключение

В проекте была реализована система для оценки степени синхронизации нотных записей и музыки в реальном времени.

В ходе работы были проанализированы алгоритмы определения частоты основного тона, алгоритмы сопоставления временных последовательностей. На основе анализа были выбраны оптимальные методы и алгоритмы с модификацией. Также был разработан метод для оценки степени синхронизации на основе нескольких агентов, работающих параллельно.

Далее было реализовано кроссплатформенно ПО на основе разработанного подхода компиляции под разные целевые платформы.

Разработанные модифицированные алгоритмы, а также методика были исследованы по различным объективным и субъективным критериям.

# Список использованных источников

* + 1. The free encyclopedia. <http://en.wikipedia.org>
    2. Arshia Cont. A Coupled Duration-Focused Architecture for Real-Time Music-to-Score Alignment. IEEE Computer Society – June 2010, 32p.
    3. Tristan Jehan. Music Signal Parameter Estimation. CNMAT Berkeley, USA. 1997
    4. David Gerhard. Pitch Extraction and Fundamental Frequency: History and Current Techniques. Department of Computer Science. University of Regina - November, 2003, 23 p
    5. YIN, a fundamental frequency estimator for speech and music - Alain de Cheveigne, Hideki Kawahara - 9 January 2002.
    6. Pitch detection methods review. <https://ccrma.stanford.edu/~pdelac/154/m154paper.htm#_ftn12>
    7. Dannenberg, R. B. (2000) Artificial Intelligence, Machine Learning, and Music Understanding. Proceedings of the 2000 Brazilian Symposium on Computer Music: Arquivos do Simpsio Brasileiro de Computao Musical (SBCM).
    8. Dixon, S. (2001) Automatic Extraction of Tempo and Beat From Expressive Performances. Journal of New Music Research, vol. 30(1):pp. 39–58
    9. Orio, N. and Dechelle, F. (2001) Score Following Using Spectral Analysis and Hidden
    10. Markov Models. Proceedings of the 2001 International Computer Music Conference
    11. Raphael, C., Cont, A., Schwarz, D., Litterst, G., West, K., Van Schuetterhoef, A., Good, M. and Downie, S. (2006) Score Following (Discussion). <http://www.music-ir.org/mirex2006/index.php?title=Score_Following>
    12. Vercoe, B. L. and Puckette, M. S. (1985) Synthetic Rehearsal: Training the Synthetic
    13. Performer. Proceedings of the 1985 International Computer Music Conference.
    14. Rabiner, L. R. (1989) A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, vol. 77(2): стр. 257–286.
    15. И.А. Кочуркин – Универсальный подход разработки клиентской и серверной частей веб-приложений для обработки изображений. Сборник статей 2 Международной научно-практической конференции, 2013. Стр. 76-85.

# Приложение А

## Тестовые мелодии

Twinkle Twinkle Little Star



All I Ask of You





# Приложение Б

## Вероятности СММ на примере Twinkle Twinkle Little Star

N, M, A, B, π and V (согласно нотации Рабинера (1989)):

N = Количество состояний = 15

M = Количество возможных наблюдений = 12

A = Вероятности переходов

B = Вероятности наблюдений

π = Исходные вероятности

V = возможные наблюдения = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 0 | 0.0 | 0.45 | 0.1 | 0.45 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | 0.0 | 0.0 | 0.1 | 0.81 | 0.0 | 0.05 | 0.0 | 0.01 | 0.0 | 0.01 | 0.0 | 0.01 | 0.0 | 0.01 | 0.0 |
| 2 | 0.0 | 0.0 | 0.0 | 0.45 | 0.1 | 0.45 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.82 | 0.0 | 0.05 | 0.0 | 0.01 | 0.0 | 0.01 | 0.0 | 0.01 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.45 | 0.1 | 0.45 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.83 | 0.0 | 0.05 | 0.0 | 0.01 | 0.0 | 0.01 | 0.0 |
| 6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.45 | 0.1 | 0.45 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.84 | 0.0 | 0.05 | 0.0 | 0.01 | 0.0 |
| 8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.45 | 0.1 | 0.45 | 0.0 | 0.0 | 0.0 |
| 9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.85 | 0.0 | 0.05 | 0.0 |
| 10 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.45 | 0.1 | 0.45 | 0.0 |
| 11 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.9 |
| 12 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.5 | 0.5 |
| 13 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| 14 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

Таблица 1 – Вероятности переходов из одного состояния в другое.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 0.0 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.09 |
| 1 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 0.0 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.09 |
| 3 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.0 | 0.091 | 0.091 | 0.091 | 0.09 |
| 5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 6 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.0 | 0.091 | 0.091 | 0.091 | 0.09 |
| 7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 8 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.0 | 0.091 | 0.09 |
| 9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 10 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.0 | 0.091 | 0.09 |
| 11 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 12 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.091 | 0.0 | 0.091 | 0.091 | 0.091 | 0.09 |
| 13 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 14 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 | 0.083 |

Таблица 2 – Вероятности наблюдений.

|  |  |
| --- | --- |
| Состояние | Вероятность |
| 0 | 0.1 |
| 1 | 0.75 |
| 2 | 0.0 |
| 3 | 0.05 |
| 4 | 0.0 |
| 5 | 0.04 |
| 6 | 0.0 |
| 7 | 0.03 |
| 8 | 0.0 |
| 9 | 0.02 |
| 10 | 0.0 |
| 11 | 0.007 |
| 12 | 0.0 |
| 13 | 0.003 |
| 14 | 0.0 |

Таблица 3 – Исходные вероятности.

# Приложение В

Следующие диаграммы были взяты из [13]. При этом выбирался орнамент, наиболее соответствующий музыкальному стилю (хотя это и субъективный процесс):



Рисунок 1 – Короткий форшлаг. Быстроиграемая нота, добавленная до основной.



Рисунок 2 – Форшлаг. Короткий форшлаг. Быстроиграемая нота, добавленная до основной.



Рисунок 4 – Инвертированный мордент. Быстро играемые ноты, играемые последовательно.



Рисунок 5 – Глиссандо. Быстро играемые вверх или вниз ноты.



Рисунок 6 – Трель. Две периодически повторяющиеся ноты.

# Приложение Г

<?xml version="1.0" encoding="utf-8" standalone="no"?>

<!DOCTYPE

score-partwise PUBLIC "-//Recordare//DTD MusicXML 2.0 Partwise//EN"

"http://www.musicxml.org/dtds/partwise.dtd">

<score-partwise version="2.0">

<part-list>

<score-part id="P1">

<part-name>Music</part-name>

</score-part>

</part-list>

<part id="P1">

<measure number="1">

<attributes>

<divisions>1</divisions>

<key>

<fifths>0</fifths>

</key>

<time>

<beats>4</beats>

<beat-type>4</beat-type>

 </time>

<clef>

<sign>G</sign>

<line>2</line>

</clef>

</attributes>

<note>

<pitch>

<step>A</step>

<octave>5</octave>

</pitch>

<duration>6</duration>

<voice>1</voice>

<type>half</type>

<dot></dot>

</note>

<note>

<pitch>

<step>C</step>

<octave>6</octave>

</pitch>

<duration>2</duration>

<voice>1</voice>

<type>quarter</type>

</note>

</measure>

</part>

</score-partwise>