The Prisoner's Sonata:
Modeling Musical Improvisation with Game Theory

Caroline Marcks, Andrew Mendelsohn, and Jayme Woogerd

1 Introduction

What problem did we set out to solve?

1. Modeling interplay between musicians
2. Build a tool to help people generate music this way
3. Another thesis statement?

2 Background

2.1 Music and Improvisation

In *A Model of Performance, Interaction, and Improvisation*, Paul Hudak outlines a formal model of musical performance, interaction and improvisation based on the idea that a full musical performance can be understood as a set of complex, interrelated interactions.

To motivate this model, he observed that in any performance there exists an interaction between a player and himself: throughout the performance, a good musician will continuously make adjustments to his playing based on what he is hearing from his own instrument. Likewise, in an ensemble or orchestral performance, a player is necessarily affected by the performance of each of the other players in the ensemble. Finally, each player is working off his or her interpretation of the musical score and, in the case where the players are improvising, the produced performance may actually deviate wildly from what is given in the score.

In the paper, these interactions are termed *mutually recursive processes* where "the recursion captures feedback [and] mutual recursion captures the interaction between players..."

In concrete terms, we define these relationships of interaction algebraically:

```
r = instr(player s r)
```

where *r* is the realization, the music actually produced by a player, and *s* is the score.

With this general model for performance and interaction in mind, Hudak asks us to further consider improvisational settings, where he notes, "although the goal of those involved in improvisation in generally one of cooperation, there is also a certain amount of conflict." Given this natural relationship of conflict and cooperation between players in an improvisational scenario and the mathematical model we can use to frame it, he suggests the application of *game theory* where "engaged processes can be viewed as players in a game, where currency is manifested as aspects of musical aesthetics, and the rules relate to control of such aesthetics.

2.2 Game Theory

Game theory is the study of strategic decision making. It is used to model interactions between agents whose decisions affect each other's welfare.

Before one can formally model a game, a number of things must be defined in the context of that game. The word "game" refers not to a something a group of kids play at recess, but to any situation with more than two decision-makers. Those decision-makers are called the players, and there must be a set of rules that discuss the set of legal moves each player can make. Those each move must be defined in terms of how it effects the state of the game. In picking what moves to make, each player must adhere their own fixed and well defined strategy that can be algebraically formalized. There must be strict definitions for what information is available in the game to be used by the strategies, both shared amongst the players and not. Separately from payoff, there must be a measure of success for each player by the end of the game, called that player's payoff. All of these things together lead to a well defined game that can be modeled and reasoned about.

Much of the use of game theory is centered around the development of sound and optimal strategies. An optimal strategy is one that will always lead the player to their best possible payoff. But how can we begin to reason about the process of reaching such a payoff? The answer lies in game trees. Game trees represent every possible sequence of moves from a starting game state to an ending game state. The root of the game tree is the starting state, and each branch represents a possible move for a player or chance from an external event to change the game state. The leaf nodes represent the "game over" states and have a payoff for each player associated with them. The best strategies are those that themselves have a concept of a game tree, and traverse it in such a way that they know they will reach optimal payoff nodes.

How does all of this fit into the context of a real game? Let us consider a formal treatment of the game TicTacToe. The decision makers in TicTacToe are the characters 'X', and 'O'. They must alternate moves, and place their mark on a 3 by 3 grid, in a previously unoccupied cell. When there are three cells in a row occupied by the same player (horizontally, vertically, or diagonally), that player wins and the game is over. If the board fills up before this can happen, there is a draw. The players share the knowledge of where they have each moved on the board. but nothing further. The payoffs are fixed to be a 1 for the winning player, -1 for the losing player, and 0 for each player in the case of a draw. Now let us see what this might look like in the first few levels of the game tree:

–TicTacToe visual here

As you can see, the first row models player X making a move, and O's moves in the second row branch off from X's. In extensive games where players each make multiple moves, these game trees can grow very rapidly. TicTacToe, for example has 25,000+ nodes in it's game tree, even when eliminating nodes where we can through rotational symmetry.

But how do game trees work when more than one player must make a decision at a time? This is where it is necessary to have strict and defined knowledge boundaries for the players. Simultaneous games are modeled as a layer of strictness on top of alternating games, like TicTacToe. Players must still make moves one by one, but the information of that choice does not get shared until all players have made their choices.

–should I go into RPS example here?

–possible transition paragraph here.

2.3 Hagl

Traditionally, game theory has been used to mathematically determine the best course of action for any given situation. However, the best course of action is often the selfish one, and requires much computational effort to calculate on the fly. A subdivision of game theory, called experimental game theory, examines games in the light of competitive experimentation. This is particularly relevant to games where sub-optimal strategies might yield a better net result. In Improvise, we are interested in a yielding music that is good for both players, rather than just one at the cost of the other.

Hagl is a DSL embedded in Haskell that allows for easy and modular definitions of games. In Hagl, a game is an instance of the Game type class:

```
class GameTree (TreeType g) => Game g where
    type TreeType g :: * -> * -> *
    type State g
    type Move g
```

```
gameTree :: g -> (TreeType g) (State g) (Move g)
```

TreeType, State, and Move are associated types that must be defined in terms of the game. The only function that must be provided is the gameTree function, that takes a game, and builds it's whole game tree. Hagl provides a number of operations to then interact with the game, which all involve examination of the built game tree, which is why it is a requirement that the TreeType also is an instance of his GameTree class, which contains operations for examining nodes and the moves that branch off from every node.

Although it is possible to write your own instance of GameTree, Hagl provides two generic representations of GameTrees: Continuous, and Discrete. Intermediate nodes have a state and list of outbound edges associated with them, and terminal nodes are payoff nodes, that contain a list of Floats, which are the payoffs for each player in the order the players were passed into the game.

The game tree edges, as mentioned in the introduction to game theory, must have a concept of payoff associated with them. In Hagl, this should be represented through

Players in Hagl are represented as a simple data type:

```
data Player g = forall s. Player Name s (Strategy s g)
              | Name ::: Strategy () g
```

Here the Name is just a String, and they must be associated with a strategy, and possibly maintain personal information within the universally quantified type variable, s.

–struggling to talk intelligently about the strategy type – how do I explain this:

```
data StratM s g a = StratM { unS :: StateT s (ExecM g) a }
type Strategy s g = StratM s g (Move g)
```

The execution of a game happens through the evalGame call:

```
evalGame :: Game g => g -> [Player g] -> ExecM g a -> IO a
```

ExecM is the game execution monad, but in this context, it's simply a sequence of operations to evaluate. This can be running through the entire game with the 'finish' call, or stepping a number of times (step ¿¿ step ¿¿ step, etc.).

–was weird to try to fit in information about ByPlayer and the other list operations here. can we just put them in as we need them?

3 Representational Concepts

3.1 Music as a Game

Given this general game theoretic framework, how do we then map the fundamental components of a game to aspects of an improvisational performance of two or more musicians?

The first critical component is the concept of moves or decisions available to each player, and the rules that specify those moves for all possible game states. For a musical game, moves take the form of *fixed-duration, time-stamped musical events*. Rather than thinking of a player's full performance as a sequence of notes, we imagine it as a sequence of musical events, that is, a stream of his or her decisions of what and how to play at each time-stamp.

Events must necessarily be both time-stamped and of a fixed-duration to ensure that players make their decisions of what to play for each event *at the same time*. Notes in the musical sense are associated with a given duration, but consider a game in which one player begins playing a whole note and the other a quarter: when should we define the next decision point at which the player's can make their next moves? When the second player is ready to make another move, the first still has three more beats to play. Thus, we break notes into musical events of fixed-duration and allow a player to choose to extend the pitch of the previous event if he or she desires to play a note of a longer duration.

In his paper, Hudak notes that the formal model of musical interaction" is limited to controlling an instrument's sound, for the purposes of realizing fundamental parameters such as pitch in addition to more subtle issues of articulation, dynamics and phrasing." Indeed, in our simple implementation, we focus solely on controlling pitch, ignoring the more subtle parameters and keeping the players instruments fixed.

Second, the currency or payoff of the game, used to measure a player's success in the performance, translates to a player's notion of musical aesthetic. This is a measure of how good the musician considers his own performance sounds, given the actions of all the other players. This preference is unique to each player and may be defined along any axis of the musical design space.

The realm of musical aesthetic is practically infinite, and a full discussion of music theory of this depth is well beyond the scope of this paper. However, we decided to model an extremely simple idea of one axis upon which a player may judge the sound of his or her performance, that of pitch intervals - the relative distance on the scale between two notes.

Finally, each player in a musical game must implement his or her own strategy,which determines how a he or she will move in any given game state. In a musical improvisational game, a strategy can be as simple as a player adhering to his or her given score, never improvising at all. Alternatively, a strategy can

be more sophisticated, taking into account past or anticipated payoffs, a player's own past moves or those of another player, or some other musical strategy with the only restriction that any move generated by the strategy fall within those allowed by the rules of the game.

Lastly, for each individual, the goal of the game is to maximize his or her own payoff. However, we note that unlike zero-sum games, which are purely competitive, a musical game is more cooperative in nature. Indeed, there exists *some* competition between players, (Hudak imagines two soloists "vying for attention"), however, intuitively, there is a point at which trying to make the other player "sound bad" (i.e. reduce his payoff) has an equally deleterious effect on one's own performance. Therefore, the best outcome for each player individually is likely to be one in which the sum of their collective payoffs is also maximized.

For example, in the simplest case, we imagine two horn players soloing simultaneously in a jazz quintet. Using Hudak's algebraic formulation, the relationship between these two players is a pair of mutually recursive functions:

```
r1 = instr1(player1 s1 r1 r2)
r2 = instr2(player2 s2 r1 r2)
```

In terms of the game, *r* refers to each player's realization, the sequence of time-stamped musical events, *s* refers to each player's score (also a sequence of time-stamped musical events) and *player1* and *player2* stand for strategies employed by the given player. Legal moves at any given point in the game tree are derived from a player's score and realization and payoffs are calculated from realizations and information about the players' preferences.

3.2 The Game Tree

As previously mentioned, each branch of a game tree represents a move of one player or the chance of something effecting the game state. We eliminate the possibility of chance in this game, as the only things that effect state in this game are the player's own choices. So, every branch then represents a choice the performer has made to stick to or deviate from the score. However, moves in this game are made at the same time, so we must not modify the state associated with the game until both players have made their move.

We can now model how the state changes from node to node, but how are players allowed to pick their moves? Technically, there are an infinite number of possible deviations from a score. Since it's not feasible to include an infinite number of states in the game tree, let's first limit it to the 4 octaves on either side of middle C, and further just to notes in the western scale – a total of 96 notes. When considering a score with two instrumentalists who get to make two moves each, we're already up to over 85 million nodes in the game tree, as

the tree grows exponentially. Because of this, it is very important to further limit the possible moves as well as construct strategies that take advantage of lazy evaluation and do not explore all nodes of the tree. Because the effects of changing the legal set of moves in a game are so drastic (on a computational level), this is something dynamic and changeable in an instance of the Improvise game.

4 Improvise: Our Implementation

4.1 Static Components

In order to implement music as a playable game, we must implement the aforementioned components of games: moves, payoffs, and strategies. The choice made here define the framework for the game itself, and therefore, the range of music that can be produced from it.

The first important piece is the MusicMv:

```
data MusicMv = Begin  Pitch
             | Extend Pitch
             | Rest
```

This defines a move as either a Begin of a Pitch, Extend of a Pitch, or a Rest. Each represents a musical event of the games defined atomic duration. The distinction between begin and extend, however, is somewhat subtle, as is the reasoning behind giving Extend a pitch at all. A Begin represents the onset or attack of the given pitch, whereas an Extend is a continuation of the previous note without a distinct onset. The difference lies in the following example: Given a sequence of moves [Begin p, Begin p], the sound produced would be a note of length one unit followed by another note of length one. There will be an audible separation of the two notes and a distinct beginning of the second immediately following the termination of the first. In contrast, the move sequence [Begin p, Extend p], would produce a note of twice the duration with no audible onset beyond the first. The notes will sound as one continuous event lasting two units. A Rest represents no audible sound, simply one unit of silence.

It is necessary in our implementation for an Extend to directly follow either a Begin or another Extend. In both cases, the Extension of the preceding note must contain the same pitch. It is unclear then why Extend should be given a pitch at all, as it should be simple enough to look back through the previous moves until the most recent begin, to which the Extend is being applied, is found. We found this to be slightly impractical later on when attempting to write strategies and payoffs. The storage of the Pitch must be weighed against the cost in time of repeatedly looking back. This decision also lifts the burden

from the programmer when writing payoffs and strategies, a burden we look to minimize overall.

1. anything data related
   (a) MusicMv - make sure to explain why we're using Extend Pitch rather than just Extend
   (b) SingularScore (Performer)
   (c) RealizationState
2. functions related to data – who, end, registerMove
3. representation of the tree – Discrete s mv, gameTree generation function

4.2 Dynamic Components

1. payoff (example)
2. list of possible moves
3. talk about how this is wrapped up in our Improvise data type with the starting state
4. strategies (example)

5 Outcomes

1. Case study: Mary Had A Little Lamb: interval payoffs, playable function based on range, preferences
2. Case study

6 Conclusion

7 Future Work

1. Perhaps your paper can say how such functions would be written and whether it is computationally feasible to evaluate them? (from Norman's feedback)
2. Continuous game?