Playing with the Strings: Designing Puppitor as an Acting Interface for Digital Games

Nic Junius, Michael Mateas, Noah Wardrip-Fruin, Elin Carstensdottir
University of California, Santa Cruz
1156 High Street
Santa Cruz, California 95064
njunius@ucsc.edu, michaelm@soe.ucsc.edu, nwardrip@ucsc.edu, ecarsten@ucsc.edu

Abstract

Interactive drama has focused on how to allow the player agency in influencing their experience through plot action. Interactive drama's preoccupation with changing plot structure bears little resemblance to theater's emphasis on character expression and dramatic play. Dramatic play allows the player to embody the character through actions, focusing on how characters express themselves and react rather than on how or even if that impacts the overarching sequence of events. In this paper, we present *Puppitor*, a system for character expression of emotion for interactive storytelling built using acting practice and fighting games as the foundation for its core design and describe its usage in conjunction with Ren'Py to facilitate a novel interactive narrative experience as a case study.

Introduction

Interactive drama has sought to tackle the question of how to allow the audience to become a player in the narrative by drawing heavily from theatrical practices and theory. For the systems that explicitly describe themselves using this term, the ability for a player to change the plot structure and direction through their choices (plot action) is a central component of turning an audience member into a player. However, when looking at the way theater itself is created, particularly practices that surround acting and its relationship to a script, this preoccupation with changing plot structure bears little resemblance to the art of creating drama, described as embodying and giving life to a literary character through actions on stage (stage action) (Levin and Levin 2002). Plot action is concerned with the events of a story, what happens, who is involved, what order do the events happen in. Stage action is concerned with the details of the way those events are enacted, how does a character say something, what is the reaction to a movement, how does a character convey a feeling. Interactive drama asks *what* characters should do in a story. For theater, *how* characters act in a story is equally important.

Therefore, the question of how characters express themselves and how interaction with characters is conducted by the player, is significant to any study and development of AI characters in interactive drama systems. Character expression and reaction is central to the feedback players use to determine how to react and reason about AI characters and shapes their expectations for the interaction. Further, the interface and/or control scheme the players use to communicate their intentions back to an AI character shapes their understanding and experience (Sali et al. 2010).

Many existing interactive drama systems' approaches rest on the assumption that constructing or altering plot structure based on actions and events taking place during a simulation is the way to include a player in a drama. For example, Façade enforces a degree of plot structure using its underlying reactive planner written in ABL (Mateas and Stern 2002) to create a coherent and focused narrative while giving the player a wide array of choice in how they act. That is not to say that these systems lack expressivity for their characters. Versu (Evans and Short 2014) takes a more distributed approach to storytelling by allowing characters', whether AI or player controlled, pursuit of their own goals to drive conflict in the plot rather than enforcing a more structured experience and uses character's subtle expressions as a core part of interaction.

While these systems support expressivity for their characters, their emphasis and focus on altering the plot tends to turn this expressivity into another means for impacting the plot progression rather than facilitating the exploration of a story through dramatic play. Dramatic play allows the player to embody the character through actions and in doing so focus on how the characters express their emotional reaction rather than how or even if that impacts the overarching sequence of events. While both approaches have merit, the exploration of player expressivity and play with AI characters in the context of dramatic play has been comparatively under-explored.

In this paper, we present *Puppitor*, a system for character expression of emotion for interactive storytelling. Puppitor centers expressivity and dramatic play to allow players to focus on interaction with AI characters rather than focusing on plot action. Puppitor is a computational caricature (Smith and Mateas 2011) of physical theatrical acting practice, that allows players to control the characters' physicality in a narrative space similarly to how they might control fighting game characters in a combat context. Built on existing acting practice (Levin and Levin 2002; Bogart and Landau

2005; Rimer, Yamazaki et al. 1984), Puppitor provides an embedded domain specific language for specifying character rules for how performing actions convey emotion. In this paper, we describe Puppitor in detail, and showcase how it enables varied dramatic character expression through a case study of the visual novel *Tracks in Snow*.

Related Work

Interactive Drama

Interactive drama traces its roots back to Brenda Laurel's dissertation (Laurel 1986) and the development of interactive Aristotelian dramatic theory through the Oz project (Mateas 1999) and *Façade* (Mateas 2001; Mateas and Stern 2002, 2005). Foundational to interactive drama is the idea that plot must react to a player's actions. In pursuit of more simulation oriented approaches, psychology and sociology were leveraged to develop and augment interactive drama systems (Si, Marsella, and Pynadath 2005; McCoy, Mateas, and Wardrip-Fruin 2009; McCoy and Mateas 2009; Evans and Short 2014) such as *Comme il Faut* which draws from Goffman's social dramaturgy to guide its creation of modular social exchanges (McCoy and Mateas 2009). Other systems are not as steeped in the Aristotelian approach and draw from a variety of sources (Szilas 2003; Cavazza et al. 2007; El-Nasr 2007) such as the story engine from *Madame Bovary on the Holodeck* explicitly using Gustav Flaubert's inventory of feelings to direct character goals (Cavazza et al. 2007). None of these systems, however, refute the core assumption of the Aristotelian approach: that changing the plot is core to interactive storytelling.

These interactive drama systems commonly have some sort of emotional model as part of their character definitions, from *Madame Bovary*'s inventory of feelings (Cavazza et al. 2007) to Versu's emotional states based on Ekman's typology (Evans and Short 2014) to *Mirage*'s usage of *Hap* supported by an emotion model similar to FLAME (El-Nasr 2007; El-Nasr, Yen, and Ioerger 2000). When compared to a system specifically designed as an emotional model, such as GAMYGDALA (Popescu, Broekens, and van Someren 2014), the distinction between an emotional model and performance model, such as Puppitor, becomes clear. First, GAMYGDALA uses psychology as its theoretical foundation while Puppitor uses theater. GAMYGDALA implements a specific model of emotion, OCC (Ortony, Clore, and Collins 1990), and using the system requires use of that particular emotional model whereas Puppitor is designed for use with multiple different theories within its framework beyond the Stanislavsky, Nō, and Viewpoints domain described in this paper.

Acting Practices

The primary influences for Puppitor's design are Stanislavsky's Method of Physical Actions (Levin and Levin 2002), Viewpoints (Bogart and Landau 2005), and Nō theater (Rimer, Yamazaki et al. 1984). Each provides a unique perspective on how to express emotions and character through physicality. For a more detailed discussion for how these practices relate to characters in digital games see (Junius, Mateas, and Wardrip-Fruin 2019). We summarize acting practice influences for Puppitor's design specifically below.

Over the course of his career, Stanislavsky iterated his acting philosophy numerous times, the most well known of which is the Method with its emphasis on analyzing character objectives (Levin and Levin 2002). The Method of Physical Actions, a later iteration, emphasizes the connection through physical action between actors on stage (Levin and Levin 2002). Puppitor draws from the latter to model the way gesture conveys feeling. Central to the Method of Physical Actions is the idea that actions on stage are all performed with some amount of energy flowing from somewhere within an actor towards someone else on stage. Stanislavsky calls these actions and their changes over the course of a sequence of performances an actor's adjustments (Levin and Levin 2002).

Like the Method of Physical Actions, the Viewpoints are a reaction to the internal focus of Stanislavsky's original Method, stemming from innovations in choreography in the 1960s and 70s (Bogart and Landau 2005). Anne Bogart and Tina Landau define the Viewpoints as a set of names given to certain principles of movement through time and space, constituting a language for talking about what happens on stage (Bogart and Landau 2005). They use nine Physical Viewpoints separated into the categories of Time and Space. Puppitor draws from the viewpoints to define the way gestures can be held and repeated by players.

In his writings on Nō theater, Zeami focuses heavily on the audience's experience of a performance and how an actor's physicality can emphasize elements of a script but the script importantly provides context for that physicality (Rimer, Yamazaki et al. 1984). He describes this as "communicat[ing] first by hearing, then by sight" and describes how actions preceding their context will have diminished readability and weight (Rimer, Yamazaki et al. 1984). Rather, he suggests that a person's intentions give way to their behavior and that the context the words of a script provide ground those actions (Rimer, Yamazaki et al. 1984). Importantly, the way an actor speaks and gestures should align both with each other and with the text. Puppitor uses this inspiration to guide the creation of its set of actions to be expressive and fit a wide range of contexts.

Applications of Theater in Computational Media

The aforementioned acting practices have been used in conjunction with prior computational media work. Stanislavsky's volitional objectives have been used as inspiration for the character behavior system in *Mirage* (El-Nasr 2007) and his active analysis technique (an evolution of volitional objectives) has been used as part of a framework for crowd sourced social simulation training (Feng et al. 2016). The Viewpoints have been used as part of allowing AI systems to interpret human gestures (Jacob, Zook, and Magerko 2013). While both Puppitor and the Viewpoints AI use Bogart and Landau's Viewpoints as their foundation, each system uses the acting practice for different ends. The Viewpoints are rules for interpreting live human gestures in the Viewpoints AI (Jacob, Zook, and Magerko 2013). In con-

trast Puppitor uses the Viewpoints as part of the design of the interface on a more standard game controller (the keyboard). The use of masks in Nō theater as well as Stanislavsky's Method have been noted as inspirational for work in transformative play (Tanenbaum 2011; Tanenbaum and Tanenbaum 2015). While all of this work in computation does draw from similar acting practices as the ones underpinning Puppitor's design, their primary goals are not character expressivity and they thus draw from different aspects of each practice.

Translating from Theater to Computation

Puppitor is the synthesis and translation of the three aforementioned acting practices, the Method of Physical Actions, the Viewpoints, and Nō, into a compuational system built around the categories of *actions* and *modifiers*. Actions are major changes in a character's physicality which directly carry emotional weight. Modifiers are small changes applied to actions which alter the relationship of actions to the expression of emotions. Puppitor's architectural decisions and connection between actions and expression mostly draw from the Method of Physical actions and Viewpoints while the way the system relates to narrative and its approach to using text to contextualize its library of actions draws heavily from Nō.

The flow of energy and the concept of adjustments from the Method of Physical Actions are what became actions and modifiers respectively in Puppitor. The actions are based on the three energy states that Stanislavsky describes: unrestricted flow of energy between partners (Levin and Levin 2002) became *open* flow in Puppitor, restraining energy towards a partner (Levin and Levin 2002) became *closed* flow, and pushing a partner away with energy (Levin and Levin 2002) became projected energy. Adjustments became the general idea of modifiers, though the specific modifiers implemented in our case study's domain come from the Viewpoints.

As Puppitor doesn't have a central component of space, the Viewpoints of Time were the focus for its design. All four, tempo; duration; kinesthetic response; and repetition (Bogart and Landau 2005), exist in Puppitor in some form but tempo, duration, and repetition are the ones most present. Tempo became a set of modifiers: tempo up and tempo down. Duration became the fundamental method of interaction, a player performing an action and modifier for any desired length of time by pressing and holding a button down. Repetition, while less singular in its presence, is a necessary part of building the core interaction model around a limited set of actions and modifiers available. How all of these actions and modifiers relate to text and other non character elements draws from Nō's views on how scripts contextualize action on stage.

Unlike The Method of Physical Actions and The Viewpoints, Puppitor's usage of Nō elements is less focused on the structure of the system itself and instead on how the system fits with narrative text. Zeami views it as part of the narrative text's job to contextualize the actions of a performance (Rimer, Yamazaki et al. 1984). Puppitor, with its necessarily limited set of actions and modifiers, similarly relies on the text surrounding it to add further specificity and weight to the performance of each action. *Tracks in Snow*'s implementation of Puppitor adds an element of line reads to the way a character performs by changing the typography of the displayed line based on the expressed emotion of a character when they began saying the line. This in turn further grounds the physical actions the character is taking and allows players flexibility to play with contrasting and emphasizing the emotional weight of any given line in a scene.

Fighting Game Characters

Theater practice does not provide a sufficient guide for developing the computational underpinnings of a system for character expression. There is, however, an existing model of computational physicality that shares some of the same values as Puppitor: Fighting games.

Fighting games are an abstraction of martial arts (Miller 2020). At their core, fighting games rely on tracking a character's state over time for each move they do using frame data (Jett 2012): the information determining how long a move takes to be able to damage an opponent, how long it can damage an opponent, and how vulnerable it makes a character when they miss, hit, or are blocked. This abstraction matches the level of detail to what was desired for Puppitor's relationship to theatrical acting as fighting games tie a single action to a button press while allowing for those actions to be modified, such as pressing a button making a character punch but adding a specific motion allowing them to throw a fireball instead.

Fighting game frame data usually specifies a sequence of states: startup, active, recovery, roughly simulating the physics of throwing a punch or kick. Frame data became the basis for translating the relatively coarse actions and modifiers in Puppitor into more nuanced expressions of emotion and going beyond each action being tied to a single emotion. Puppitor allows for fluidly updating all of its emotional affects with each performed action, roughly simulating Stanislavsky's description of energy flowing out of internal movements.

Puppitor does not seek to replicate the exact technical complexity of inputs found in fighting games like *Street Fighter IV* (Capcom 2008), *Tekken 7* (Bandai Namco 2015), or *Under Night In-Birth* (French Bread and Ecole Software 2012) though it still takes direct inspiration for aspects of its interface design. *Tekken* maps each of its four buttons onto a specific body part and action combination: left punch, right punch, left kick, and right kick, then lets each of those buttons be modified based on the direction of movement. For example standing left punch is a completely different kind of punch than forward left punch even if they both rely on the character's left arm. In contrast, *Under Night* simply labels its buttons A, B, and C broadly mapped onto increasing reach and damage, with A being the shortest, fastest moves and C being the longest, slowest ones. What exactly A, B, and C correspond to in *Under Night* is entirely character specific, one character's A might be a short kick, a punch with a sword hilt for another character. Additionally, many moves in *Under Night* require more complex individual inputs, when compared to *Tekken*, combined with a button

press to say, throw a projectile. Puppitor's interface uses the broader connection between button and action from *Under Night*, mapping a button to a general category of action, such as mapping N to the *open flow* action. The modifier keys act more closely to directions in *Tekken* where combining *open flow* with *tempo up* is an action with some different characteristics than the neutral version.

System Architecture

Puppitor is broken up into two primary modules, the *Action Key Map* and the *Affecter*. First, the Action Key Map translates player or AI input into an action and modifier for a character to express. The Affecter then uses this information to update that character's *Affect Vector* (their emotional state) based on the *Character Affect Rules*, specified in a JSON file. The Character Affect Rules describe the way every action taken and modifier applied will change values in an Affect Vector each update cycle. The action and modifier expressed out of the Action Key Map and the values stored in the Affect Vector can then be used outside of Puppitor's core modules to drive character animations, change facial expressions, alter musical scores, and any other methods of feedback a designer may want. Puppitor is intended to be a library and used beyond the visual novel developed alongside it. Therefore, the specifics of how the system's outputs are used to alter visuals, text, and audio is necessarily project dependent.

As part of implementing feedback in the visual novel *Tracks in Snow* (see the case study), the Python version of Puppitor includes an animation system, designed for use with standard frame by frame sprite animation. This system not only enables dynamic visuals for the system, it serves as an example of how Puppitor's output can be used as part of a storytelling experience and connected to elements beyond its own modules. Since animation pipelines are very malleable and project specific, this functionality is generally outside the scope of Puppitor's core set of features (and outside the scope of this paper), even if the system is intended to work in conjunction with animation systems.

Beyond being incorporated into external code bases and tools like Ren'Py (Rothamel 2004), Puppitor is designed to be fully customizable within its action, modifier, affect framework. The fixed parts of the system are its general categories of actions and modifiers as well as the method used to update its emotional state. All the specifics of what actions and modifiers are used, the domain of the emotional state, and the relationship between actions and emotions are all completely customizable. All examples in this paper use actions derived from Stanislavsky's Method of Physical Actions: open flow, closed flow, projected energy, and resting. Modifiers are derived from the Viewpoints: tempo up, tempo down, and neutral. The set of emotions are: joy, anger, sadness, worry, fear, and love. We want to stress that each of these sets, while heavily rooted in the acting practices inspiring Puppitor, are only one possible choice for using the system. As an embedded domain specific language, Puppitor supports arbitrary sets of actions, modifiers, and emotions.

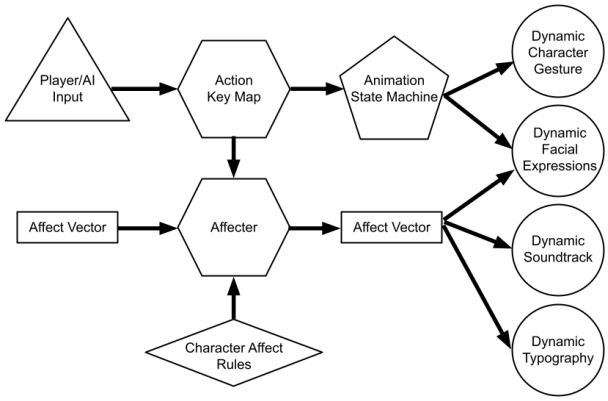

Figure 1: Puppitor translates inputs from a human or AI player (triangle) into expressive actions (circles) based on a specified ruleset (diamond) through its core modules (hexagons), which update the character's emotional state (rectangles) and serve as inputs to control character animations (pentagon).

Action Key Map

The Action Key Map module is where all of the direct input to the system (and by extension a character) is processed. Everything in the module revolves around the general categories of *actions* and *modifiers*, the exact details of which are specified as part of the Puppitor's domain specific language, an example of which can be found in the case study. *Actions* are gestures, anything that would dramatically change the way a character holds or carries themselves. *Modifiers* on the other hand are for smaller adjustments that change the feel of those larger actions. A designer specified default action and modifier gives Puppitor a way to return a character to a baseline state from any emotional state they may find themselves in. This organization by actions and modifiers is a part of each of the three elements the Action Key Map tracks: the mapping of keyboard keys to actions (key map), the tracking of key states and their associated actions (possible states), and the expressed action and modifier for use in other Puppitor modules and outside of the library itself (prevailing states).

The key map element of this module associates Puppitor actions and modifiers with arbitrary sets of keys corresponding to inputs on a keyboard. Any number of keys may be associated with a single action or modifier with the exception of the default action and modifier, which never have inputs mapped to them as they are intended to be chosen when there is no input. As it relates to external usage, the key map is where the listeners connecting the entirety of Puppitor to the keyboard are created. The key map then allows for Puppitor's actions and modifiers to be integrated more directly into code handling player input, as the link between keyboard and meaning of the inputs is handled internally rather than requiring additional setup. For example the Stanislavsky and Viewpoints based actions and modifiers can be mapped as:

```
open flow : N
closed flow : M
```

```
projected energy : B
resting : None
tempo up : C
tempo down : Z
neutral: None
```

The possible states track the key states of the associated non-default actions and modifiers, allowing multiple keys to be held down simultaneously. To continue the example from the discussion of the key map, if both the N and M keys are pressed, open flow and closed flow will be flagged as active states.

Puppitor requires that a single action and modifier be performed at any given time which is where its final element, prevailing states, comes in. This element takes the information provided by the possible states and determines which action and modifier to select as prevailing. Importantly, Puppitor itself does not determine the priority of each action and modifier, as the prevailing states' update behavior sets a specified action or modifier to active and makes every other action or modifier to inactive. How this update behavior is integrated into external programs is up to the developer using it. For example, when both open flow and closed flow are active according to possible states, if prevailing states is first told to set open flow to active and then is told to set closed flow to active, only closed flow will be active.

Affecter

The Affecter is the core of Puppitor's emotional expression capabilities. The module is built around the idea of characters having rules that connect the actions they perform (as stored in the Action Key Map) to particular expressions of emotions. A character's emotional state is represented by a mapping of emotional affects to their corresponding floating point values (the Affect Vector). Rulesets are made up of series of entries in a JSON file that define the update value each action applies to a character's emotional state, the multiplier each modifier applies to those update values, the adjacency and edge weights of other emotions, and the equilibrium point for the emotion to return to when the default action is performed. The rules are then loaded into the Affecter itself and applied to the Affect Vector each update cycle based on the action and modifier specified by the Action Key Map.

For example, if the affects joy, anger, sadness, worry, fear, and love are a Puppitor domain's set of emotions, each will have a corresponding entry in each character's affect vector and ruleset. The values in the Affect Vector default to the equilibrium point specified in a ruleset, so an initial Affect Vector could look like:

```
joy : 0.35
anger : 0.1
sadness : 0.54
worry : 0.77
fear : 0.54
love : 0.28
```

When updating an Affect Vector, the update value of an affect is multiplied by the modifier value, then added to the corresponding affect entry before the value in the Affect Vector is clamped between a designer specified floor and ceiling value. If no buttons are held, the default action and modifier will be performed and move affect values towards their corresponding equilibrium value rather than the floor or ceiling value. Without clamping, each affects values could grow or shrink to the point it would be nearly impossible for characters to express certain emotions. Continuing the example using closed flow as the active action, the update values in joy, anger, sadness, worry, fear, and love's affect rules corresponding to each affect's closed flow value would be multiplied by 1.0 (as no modifier is being applied) then added to the Affect Vector's values. Using the below affect rule entry fragment as an example, the joy value in the above example Affect Vector would be 0.3495.

```
"joy" : {
    "actions" : {
        "resting" : -0.0003,
        "open_flow" : 0.00004,
        "closed_flow" : -0.0005,
        "projected_energy" : 0.0005
    },
    "modifiers" : {
        "tempo_up" : 1.14,
        "tempo_down" : 0.5,
        "neutral" : 1.0
    },
    "adjacent_affects" : {
        "love" : 90,
        "worry" : 10
    },
    "equilibrium_point" : 0.35
},
```

The adjacency lists in a ruleset specify a directed, weighted graph of a character's associations between emotional affects, where higher weights are stronger associations. These lists take the form of percentage values mapped to their corresponding emotional affect. Within Puppitor, the adjacency lists play a small role as part of the system's provided methods for picking the highest valued affect in a character's Affect Vector. As multiple affects can potentially reach the ceiling value, the currently expressed affect is prioritized. If the currently expressed affect is no longer one of the highest valued affects, one of its adjacent affects is randomly selected based on its weight value (a weight of zero is ignored, if all adjacencies are zero, then an unweighted selection is performed). If there are no adjacencies available to pick, a simple random choice of any of the highest valued affects is made. This selection process allows the emotional state represented in the affect vector to be converted to a discrete value, a character's prevailing affect, for use outside of Puppitor, should a single value be preferable to the entire emotional state.

Where Puppitor's adjacency lists provide much more power and expressivity is when an AI system is puppeteering a character, as they can be used to allow an AI character to interpret another character's expression of emotion (see the case study for a more in depth explanation). This interpre-

Figure 2: Chiara (left) tells Rika (right) that she'll give her space if she wants. The tones of each version of the scene pictured, angry Chiara and joyful Rika above and afraid Chiara and angry Rika below, are drastically different in the moment as well as how each scene was played before to create these two moments in *Tracks in Snow*.

tation can be accomplished by using Puppitor's adjacency lists to set the goal of an AI algorithm (for example greedy search) to be the AI's character's associated affect based on what a different character is expressing. Using the updated Affect Vector above as an example for a player character, worry is expressed as it has the highest value at 0.77. To decide how to respond, the AI character looks at the adjacency list in the worry entry of its ruleset and sees it can either respond by performing sadness or fear, each with a weight of zero, so a random choice will be made (if the selection follows the same logic as Puppitor's internals).

Expression with Puppitor: a Case Study of *Tracks in Snow*

Puppitor's primary outputs are a character's action, modifier, and prevailing affect. In *Tracks in Snow* (Junius 2021; June et al. 2021), a visual novel about two women fleeing their home during Passover and designed in conjunction with Puppitor, the system's actions and modifiers are used to allow characters to dynamically change their poses and prevailing affects are used to change characters' facial expressions. Each character's prevailing affect also changes the typography of their lines when displayed. Finally, each character has a corresponding instrument in the musical score which uses their prevailing affect to dynamically switch between tracks to further emphasize their expression of emotion.

Character Through Actions and Modifiers

The two characters in *Tracks in Snow*'s cast were designed to contrast with each other narratively, visually, and most importantly systemically using Puppitor to support and emphasize the other two aspects. Rika and Chiara share some of the complexity of fighting game characters as part of their expressiveness. Fighting game characters feel different to play based on the kinds of inputs they use as well as how their animations and frame data all connect to create an expressive character (Core-A Gaming 2016). Puppitor allows for a similar degree of characterization when combining character poses, facial expressions, all linked together using the system. Both Rika and Chiara use the same buttons on the keyboard to gesture when being played by a human, though which parts of the interface they emphasize is as much a part of their personalities as their hairstyles, poses, or ways of speaking.

Rika is a more flamboyant and mercurial character who can access most of her emotional range by changing her gestures without needing to rely on modifying any of those gestures' tempos. Her ruleset is mainly focused on using larger actions to quickly switch between expressing different emotional affects, as each action has a strong positive connection to two affects and a strong negative connection to at least two other affects. In this case briefly performing an intermediate action to change Rika's emotional state is the most reliable way of moving her towards expressing a desired affect. For example, if a player wants Rika to express anger after she has had her feet up performing open flow for some time (as in figure 2), first having her perform closed flow will reduce her expression of joy and love, as both of these affects are positively associated with the open flow action and negatively associated with the closed flow action. A few moments of being closed can then move into having her perform the projected energy action, which increases both anger and love. Since Rika performed closed flow, love's magnitude in her Affect Vector has been reduced, allowing anger to then be expressed as desired.

Chiara on the other hand is a more subdued character with a very wide worry streak who constantly has to actively overcome that worry to express other emotions. To emphasize this anxious part of her personality using Puppitor, as well as to provide a very different tactile experience from playing Rika, Chiara almost always must modify her actions to access the rest of her emotional range. Rather than having two affects strongly associated with each action by default, every unmodified action primarily increases worry. Her modifiers then are designed to be both the only way of shoving her worry aside by having each modifier and action combination correspond to a strong expression of a single affect as well as any modified action reducing worry's overall value using a negative multiplier. For example, for Chiara to even get angry (as she is in figure 2), she not only needs to perform the projected energy action, she must specifically modify it with tempo down. If she does not she will simply worry more. If she instead uses the tempo up modifier, she will become joyful as joy is the affect with the highest modifier multiplier and action value using the combination of tempo up and projected energy.

When setting up either Rika or Chiara to be performed by a human player, first their actions: *open flow*, *closed flow*, and *projected energy* are mapped to the keyboard keys of: 'N', 'M', and 'B' respectively in the key map dictionary of Action Key Map, allowing those keys to be listened for in Ren'Py's event handler. As part of the same process, their modifiers: *tempo up* and *tempo down* are mapped to the keys: 'C' and 'Z' respectively. The default action of *resting* and default modifier of *neutral* are passed into the Action Key Map. Now when any of those keys are pressed, their corresponding action in the possible states dictionary will be set to true. In the event no key is pressed, *resting* and *neutral* will be true as the respective default action and modifier. Finally, the possible states are used to determine which action and modifier are expressed in the prevailing states dictionary. The exact priority that prevailing states uses is determined as part of connecting Puppitor to other system's update functionality. In *Tracks in Snow*'s case, this priority, from highest to lowest, is: *open flow*, *closed flow*, and finally *projected energy*.

Interpretation Using Affect Adjacencies

The AI actor in *Tracks in Snow* uses a greedy search to evaluate each of the action and modifier pairs it could perform each frame and chooses the combination that will add the highest value to the affect it is given the goal of performing. By itself the search has very little ability to reason about what is put in front of it, which Puppitor's features can address. Each affect in a Puppitor ruleset can be given adjacencies to other affects within the domain, ultimately building a weighted, directed graph of how emotional affects relate to each other. In other words, the the associations a character has about each affect.

In *Tracks in Snow*, the AI actor uses the adjacency lists as primary method of allowing interpretation of what the human controlled character is doing. When the human player performs any affect, the AI receives this and then chooses an affect it wants to perform based on a weighted random selection of an affect out of the adjacencies to that received affect in its ruleset. A single adjacency will make for a fairly calm performance from the AI using this approach as it only has one association. By adding more associations, the AI will commit less to trying to perform a single affect. When used in conjunction with the weight values, the feel of these changes can create different performances. To return to the example in figure 2, if the AI is performing as Chiara and Rika just started expressing joy, Chiara's adjacencies for joy are:

```
love : 90
worry : 10
```

Approximately ninety percent of the time the AI will try to perform to express love back to Rika. The remaining approximately ten percent of the time the AI will try to perform worry back. The time it takes for emotions to change means the AI will likely never be able to actually express worry with this weighting, but it will make for a mostly stable performance with the occasional adjustment as the quick attempts to express worry pop up. With a more balanced weighting, such as:

```
love : 60
worry : 40
```

the performance will get more fidgety even with the weighting still favoring love, worry may creep up every so often and be fully expressed.

Puppitor's adjacency lists provide a way for AI characters to interpret other characters' performances through their own associations with each emotional affect as well as provide an additional way to tune an AI's performance within the domain specific language itself. By design, most of Puppitor's language focuses on the way a character expresses themselves, but with the adjacency lists, there is a built in way for characters to make choices about how they want to respond to other characters' performances as well.

Through its implementation in *Tracks in Snow*, Puppitor enables dynamic character expression either by a human or AI player. These dynamic poses and facial expressions help shape the tone of a scene alongside other sources of feedback like a reactive musical score and typography. All of this feedback is realtime and continuous. As a result of this layered approach, shadows of previous emotions can remain as scenes progress and flashes of others can surface as characters change what they are trying to express. Being a linear narrative using Puppitor, *Tracks in Snow* has an emphasis on interpretation and reflection as described by Junius et al (Junius, Kreminski, and Mateas 2021), though the loop of acting and reflecting is much more dependent on a player's desire to than built directly into the structure of the game. One significant area of improvement is enabling the AI characters to further reason about Puppitor's state to make more interesting acting choices.

Conclusion

Puppitor is a system for character expression of emotion built as a computational caricature of theater acting practices (Levin and Levin 2002; Bogart and Landau 2005; Rimer, Yamazaki et al. 1984), and allows players to control the characters' physicality in a narrative space similarly to how they might control fighting game characters in combat. Specifically, Puppitor maps actions and modifiers to keyboard inputs and based on this mapping applies rules about how emotions are expressed, all specified using an embedded domain specific language. Puppitor facilitates a wide range of expression for both human and AI players to perform characters, which we demonstrate through the case study of *Tracks in Snow*. Puppitor provides means of systematically exploring dramatic play and stage action, which the field of interactive drama has left relatively under explored in favor of plot action. Through emphasizing dramatic play, Puppitor allows us to expand the study of AI character design and interaction paradigms for interactive storytelling experiences, and more broadly AI characters as they are utilized in interactive narrative and beyond. With Puppitor we hope to both inspire further investigation of systems for supporting dramatic play as well as providing a new domain for work with character performance.

References

Bandai Namco. 2015. Tekken 7. [Arcade; PlayStation 4; Xbox One; PC Digital Download Steam], Japan: Capcom.

Bogart, A.; and Landau, T. 2005. *The Viewpoints Book: A Practical Guide to Viewpoints and Composition*. Theatre Communications Group.

Capcom. 2008. Street Fighter IV. [Arcade; PlayStation 3; Xbox 360; PC Digital Download Steam], Japan: Capcom.

Cavazza, M.; Lugrin, J.-L.; Pizzi, D.; and Charles, F. 2007. Madame bovary on the holodeck: immersive interactive storytelling. In *Proceedings of the 15th ACM international conference on Multimedia*, 651–660.

Core-A Gaming. 2016. Analysis: How to Pick a Character. https://youtu.be/AGHGEttNjyo. Video accessed on 7/31/2022.

El-Nasr, M. S. 2007. Interaction, narrative, and drama: Creating an adaptive interactive narrative using performance arts theories. *Interaction Studies*, 8(2): 209–240.

El-Nasr, M. S.; Yen, J.; and Ioerger, T. R. 2000. Flame—fuzzy logic adaptive model of emotions. *Autonomous Agents and Multi-agent systems*, 3(3): 219–257.

Evans, R.; and Short, E. 2014. Versu - a Simulationist Storytelling System. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(2): 113–130.

Feng, D.; Carstensdottir, E.; Carnicke, S. M.; El-Nasr, M. S.; and Marsella, S. 2016. An active analysis and crowd sourced approach to social training. In *International Conference on Interactive Digital Storytelling*, 156–167. Springer.

French Bread and Ecole Software. 2012. Under Night In-Birth. [Arcade; PlayStation 3; PC Digital Download Steam], Japan: Aksys Games.

Jacob, M.; Zook, A.; and Magerko, B. 2013. Viewpoints AI: Procedurally Representing and Reasoning about Gestures. In *Proceedings of DiGRA*.

Jett. 2012. Universal Fighting Game Guide: How to Read Frame Data. https://thirdpersonblog.wordpress.com/2012/04/18/universal-fighting-game-guide-how-to-read-frame-data/. Blog post accessed on 7/31/2022.

June, N.; Murray, R.; Marshall, C.; Duplantis, T.; Karth, I.; and Kreminski, M. 2021. Tracks in Snow. [PC Digital Download itch.io], USA.

Junius, N. 2021. *TRACKS IN SNOW: A DIGITAL PLAY ABOUT JUDAISM AND HOME*. Master's thesis, UNIVERSITY OF CALIFORNIA SANTA CRUZ.

Junius, N.; Kreminski, M.; and Mateas, M. 2021. There Is No Escape: Theatricality in *Hades*. In *Proceedings of the 16th International Conference on the Foundations of Digital Games*.

Junius, N.; Mateas, M.; and Wardrip-Fruin, N. 2019. Towards Expressive Input for Character Dialogue in Digital Games. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*.

Laurel, B. K. 1986. *TOWARD THE DESIGN OF A COMPUTER-BASED INTERACTIVE FANTASY SYSTEM (DRAMA, PLAYWRITING, POETICS, EXPERT SYSTEMS, THEORY)*. Ph.D. thesis, The Ohio State University.

Levin, I.; and Levin, I. 2002. *The Stanislavsky Secret*. Colorado: Meriwether Publishing.

Mateas, M. 1999. An Oz-centric review of interactive drama and believable agents. In *Artificial intelligence today*, 297–328. Springer.

Mateas, M. 2001. A preliminary poetics for interactive drama and games. *Digital Creativity*, 12(3): 140–152.

Mateas, M.; and Stern, A. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4): 39–47.

Mateas, M.; and Stern, A. 2005. Structuring Content in the Façade Interactive Drama Architecture. In *AIIDE*, 93–98.

McCoy, J.; and Mateas, M. 2009. The Computation of Self in Everyday Life: A Dramaturgical Approach for Socially Competent Agents. In *AAAI Spring Symposium: Intelligent Narrative Technologies II*, 75–82.

McCoy, J.; Mateas, M.; and Wardrip-Fruin, N. 2009. Comme il faut: A system for simulating social games between autonomous characters. *Digital Arts and Culture*.

Miller, P. 2020. Why fighting games are hard. https://pattheflip.medium.com/why-fighting-games-are-hard-7d6d423028ff. Blog post accessed on 7/31/2022.

Ortony, A.; Clore, G. L.; and Collins, A. 1990. *The cognitive structure of emotions*. Cambridge university press.

Popescu, A.; Broekens, J.; and van Someren, M. 2014. GAMYGDALA: An Emotion Engine for Games. *IEEE Transactions on Affective Computing*, 5(1): 32–44.

Rimer, J. T.; Yamazaki, M.; et al. 1984. *On the Art of the Nō Drama: The Major Treatises of Zeami; Translated by J. Thomas Rimer, Yamazaki Masakazu*. Princeton University Press.

Rothamel, T. 2004. Ren'Py. [PC Digital Download], USA.

Sali, S.; Wardrip-Fruin, N.; Dow, S.; Mateas, M.; Kurniawan, S.; Reed, A. A.; and Liu, R. 2010. Playing with words: from intuition to evaluation of game dialogue interfaces. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 179–186. ACM.

Si, M.; Marsella, S. C.; and Pynadath, D. V. 2005. Thespian: Using multi-agent fitting to craft interactive drama. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 21–28.

Smith, A. M.; and Mateas, M. 2011. Computational caricatures: Probing the game design process with ai. In *Workshops at the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Szilas, N. 2003. IDtension: a narrative engine for Interactive Drama. In *Proceedings of the technologies for interactive digital storytelling and entertainment (TIDSE) conference*, volume 3, 1–11.

Tanenbaum, T. J. 2011. Being in the story: readerly pleasure, acting theory, and performing a role. In *International Conference on Interactive Digital Storytelling*, 55–66. Springer.

Tanenbaum, T. J.; and Tanenbaum, K. 2015. Empathy and Identity in Digital Games: Towards a New Theory of Transformative Play. In *FDG*.