Game Balancing using Koopman-based Learning

Allan M. Avila[1,2], Maria Fonoberova[2], João P. Hespanha[1], Igor Mezić[1,2],
Daniel Clymer[3], Jonathan Goldstein[3], Marco A. Pravia[3] and Daniel Javorsek II[4]

***Abstract*— This paper addresses the analysis of how the outcome of a zero-sum two-player game is affected by the value of numerical parameters that are part of the game rules and/or winning criterion. This analysis aims at selecting numerical values for such parameters that lead to games that are "fair" or "balanced," in spite of the fact that the two players may have distinct attributes/capabilities. Motivated by applications of game balancing for the commercial gaming industry, our effort is focused on complex multi-agent games for which low-dimensional models in the form of differential or difference equations are not possible or not available. To overcome this challenge, we use a parameter-dependent Koopman operator to model the game evolution, which we train using an ensemble of simulation traces of the actual game. This model is subsequently used to determine values for the game parameters that optimize the appropriate game balancing criterion. The approach proposed here is illustrated and validated on a minigame derived from the StarCraft II real-time strategy game from Blizzard Entertainment.**

I. Introduction

The commercial gaming industry has a long-standing interest in maintaining game balance as balanced games are typically more entertaining, and market pressures help drive their development [1–6]. Moreover, the contemporary method for assessing and balancing games is a trial-and-error approach, thus representing an opportunity for the application of Artificial Intelligence (AI). Normally, game developers release and observe an initial configuration of the game in large scale play. Then, developers gather high-level win/loss statistics while players provide feedback about elements of the game that are overpowered or imbalanced. Finally, updates to the game are made in which elements are buffed (performance increases) or nerfed (performance decreases) to achieve game balance. To date, little quantitative modeling of game balance exists, and research on the application of AI algorithms to automating game balance assessment (formally referred to as quantitative balance analysis) is extremely limited

When designing games there are two important concepts that a game designer must keep in mind, that of game mechanics and game dynamics. Game mechanics are the technical underpinnings of the game and are essentially the game rules and objects that all players necessarily interact with in order to play. On the other hand, game dynamics are the emergent behaviors and patterns that arise from the specific way players choose to interact with and utilize the defined game mechanics. As mentioned, balancing a game is the iterative process of defining game mechanics, observing the dynamics that arise, and then tuning the mechanics to improve any undesirable dynamics that were observed. Traditionally, game developers have utilized human testing to perform this iterative procedure and for certain games, such as single-player or player versus environment (PVE) games, this method can be effective albeit expensive. However, in the case of player versus player (PVP) games, such as on-line role-playing games (RPG), the designer can not ensure that dominating strategies do not exist a priori. Thus, a PVP game may be severely unbalanced from the beginning and a considerable amount of human testing must be deployed, keeping in mind that it is not possible for a group of human testers to identify all possible strategies that could arise. Furthermore, even when an imbalance is identified finding a minimal cost and minimal effort solution may not always be so straightforward. If the game mechanics are simple then the dynamics are often easily controlled and the game can be balanced with reasonable effort. However, games can quickly become highly complex [6] involving many different types of agents, rules, terrains, vehicles, and so on. In which case the input-output relationship between mechanics and dynamics quickly becomes intractable. Thus, there is a need for automated game balancing methodologies that are capable of scaling to complex games.

Prior research efforts have utilized pattern mining

[1]University of California Santa Barbara, Santa Barbara, CA 93117
[2]AIMdyn Inc., Santa Barbara, CA 93101
[3]BAE Systems, Arlington, VA 22203
[4]U.S. Air Force, Arlington, VA 22203

*This work was supported by the Defense Advanced Research Projects Agency (DARPA). Any opinions, findings, conclusions or recommendations expressed in this article are those of the authors and do not reflect the views of DARPA, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government. This document was cleared by DARPA on October 15, 2020 and is approved for public release, distribution unlimited.

algorithms [5], Attribute search algorithms [6], and evolutionary algorithms [4, 7] to search for dominating strategies. Once dominating strategies or features are found the mechanics of the game are tuned to counteract these imbalances. While some of the previously mentioned approaches are automated balancing methodologies, others are only partially automated methods that require manual assistance. Furthermore, in many of the previously mentioned works the game mechanics are ultimately hand-tuned which can be time-consuming and lack the ability to scale to complex games. The approach this work takes is to assign a balance criterion to the game dynamics as a function of the game mechanics. Game balancing is then obtained via optimization of the mechanics to produce the desired balance criterion. In order to accomplish this, we quantify the time evolution of the game dynamics as a dynamical system where we denote the state space by $\mathbf{X}$ and the game dynamics evolve according to

$$\mathbf{x}(k+1) = F(\mathbf{x}(k), \mathbf{u}_1, \mathbf{u}_2, \boldsymbol{\theta}) \tag{1}$$

where $\mathbf{x}$ is the asset state vector, $\mathbf{u}_1, \mathbf{u}_2$ are decision vectors for both players, and $\boldsymbol{\theta}$ is the vector of tuneable game parameters representing the game mechanics. As previously mentioned, games can be quite complex which in turn would lead to a high-dimensional complex state-space representation. In order to address this difficulty, we will utilize Koopman operator methods [8, 9] to enable estimation of the governing equations for the game evolution directly from game simulation data. Figure 1 demonstrates a flowchart of the methodology. Koopman operator methods have emerged as a powerful model reduction technique for analyzing complex nonlinear dynamical systems. The Koopman group of operators induced by a dynamical system propagate the evolution of functions, referred to as observables, under the dynamics of the system. Specifically, the action of the Koopman operator group K^t on a function $\boldsymbol{\psi} : \mathbf{X} \to \mathbb{C}^m$ is $K^t f(\mathbf{x}) = \boldsymbol{\psi} \circ S^t(\mathbf{x})$, where $S^t(\mathbf{x})$ represents the flow of the dynamical system.

The Koopman operator is linear regardless of the nonlinearity of the underlying system and hence linear operator theory on Hilbert spaces can be used to analyze the evolution of observables. Specifically, the spectrum, eigenvalues/eigenfunctions, of the induced Koopman group reveal geometrical properties of the state space, [10, 11] and provide a simplified time evolution of the complex dynamics. Furthermore, there has been a plethora of algorithms developed for estimating spectral properties of the Koopman operator directly from data [12–16]. This has enabled an enormous and successful amount of data-driven modeling, analysis, and forecasting of complex systems [17–19].

II. Game Modeling Using Koopman Operators

In its original formulation, the Koopman operator was developed for the analysis of autonomous uncontrolled dynamical systems. However, the works of [20] extend the Koopman operator framework to allow for controlled systems. This achieved by first reformulating equation (1) as an equivalent dynamical system evolving on an extended space consisting of a product $\mathbf{X} \times \ell^2$ between the original state space $\mathbf{X}$ and the space of all square summable control sequences ℓ^2. The dynamics of the extended state $\hat{\mathbf{x}} = \begin{bmatrix}\mathbf{x} & \mathbf{u}\end{bmatrix}^\intercal$ are given by

$$\hat{\mathbf{x}}(k+1) = \begin{bmatrix} F(\mathbf{x}(k), \mathbf{u}_1, \mathbf{u}_2, \boldsymbol{\theta}) \\ \mathcal{S}\mathbf{u}_1(k) \\ \mathcal{S}\mathbf{u}_2(k) \end{bmatrix} \tag{2}$$

where $\mathcal{S} : \ell^2 \to \ell^2$ is the left-shift operator, $\mathcal{S}\mathbf{u}_i(k) = \mathbf{u}_i(k+1)$. We denote the set of observables by $\boldsymbol{\psi}(\hat{\mathbf{x}})$ and their functional form is restricted to be $\boldsymbol{\psi}(\hat{\mathbf{x}}) = \begin{bmatrix}\mathbf{f}(\mathbf{x}) & \mathbf{g}(\mathbf{u}_1) & \mathbf{h}(\mathbf{u}_2) & \mathbf{f}(\mathbf{x})\mathbf{g}(\mathbf{u}_1) & \mathbf{f}(\mathbf{x})\mathbf{h}(\mathbf{u}_2)\end{bmatrix}$, where $\mathbf{f} = \begin{bmatrix} f_i(\mathbf{x}), \ldots, f_{n_f}(\mathbf{x})\end{bmatrix}^\intercal$ is a vector of functions of the state, and similarly for $\mathbf{g}$ and $\mathbf{h}$. The corresponding dynamics are governed by

$$\begin{aligned} \boldsymbol{\psi}(\hat{\mathbf{x}}(k+1)) &= \mathbf{A}(\boldsymbol{\theta})\mathbf{f}(\mathbf{x}(k)) + \mathbf{B}_1(\boldsymbol{\theta})\mathbf{g}(\mathbf{u}_1(k)) \\ &+ \mathbf{B}_1(\boldsymbol{\theta})\mathbf{h}(\mathbf{u}_2(k)) + \mathbf{C}_1(\boldsymbol{\theta})\mathbf{f}(\mathbf{x}, t)\mathbf{g}(\mathbf{u}_1(k)) \\ &+ \mathbf{C}_2(\boldsymbol{\theta})\mathbf{f}(\mathbf{x}(k))\mathbf{h}(\mathbf{u}_2(k)) \end{aligned} \tag{3}$$

where the matrices $\mathbf{A}(\boldsymbol{\theta})$, $\mathbf{B}_1(\boldsymbol{\theta})$, $\mathbf{B}_2(\boldsymbol{\theta})$, $\mathbf{C}_1(\boldsymbol{\theta})$, and $\mathbf{C}_2(\boldsymbol{\theta})$ constitute a finite dimensional approximation of the Koopman operator.

The result of the above formulation is a family of Koopman operator models parameterized by the game parameters. In principle, the parameter dependency can be captured by a single Koopman model if the parameters are also lifted and the dynamics are considered to evolve on $\mathbf{X} \times \ell^2 \times \Theta$. Specifically, we can extend the state space to include the parameters $\hat{\mathbf{x}} = \begin{bmatrix}\mathbf{x} & \mathbf{u} & \boldsymbol{\theta}\end{bmatrix}^\intercal$ and the resulting Koopman representation would be

$$\begin{aligned} \boldsymbol{\psi}(\hat{\mathbf{x}}(k+1)) &= \mathbf{A}\mathbf{f}(\mathbf{x}(k)) + \mathbf{B}_1\mathbf{g}(\mathbf{u}_1(k)) \\ &+ \mathbf{B}_2\mathbf{h}(\mathbf{u}_2(k)) + \mathbf{C}_1\mathbf{f}(\mathbf{x}(k))\mathbf{g}(\mathbf{u}_1(k)) \\ &+ \mathbf{C}_2\mathbf{f}(\mathbf{x}(k))\mathbf{h}(\mathbf{u}_2(k)) + \mathbf{D}\mathbf{p}(\boldsymbol{\theta}) \\ &+ \mathbf{E}\mathbf{f}(\mathbf{x}(k))\mathbf{p}(\boldsymbol{\theta}) \end{aligned} \tag{4}$$

where $\mathbf{p}$ is a vector of functions of the game parameters. The Koopman representations (3)-(4) generated by the dynamics of equation (1) represent the most general scenario. In this formulation, the game dynamics are modeled in terms of the state, control (player inputs), and parameters (game mechanics). However, it is possible to

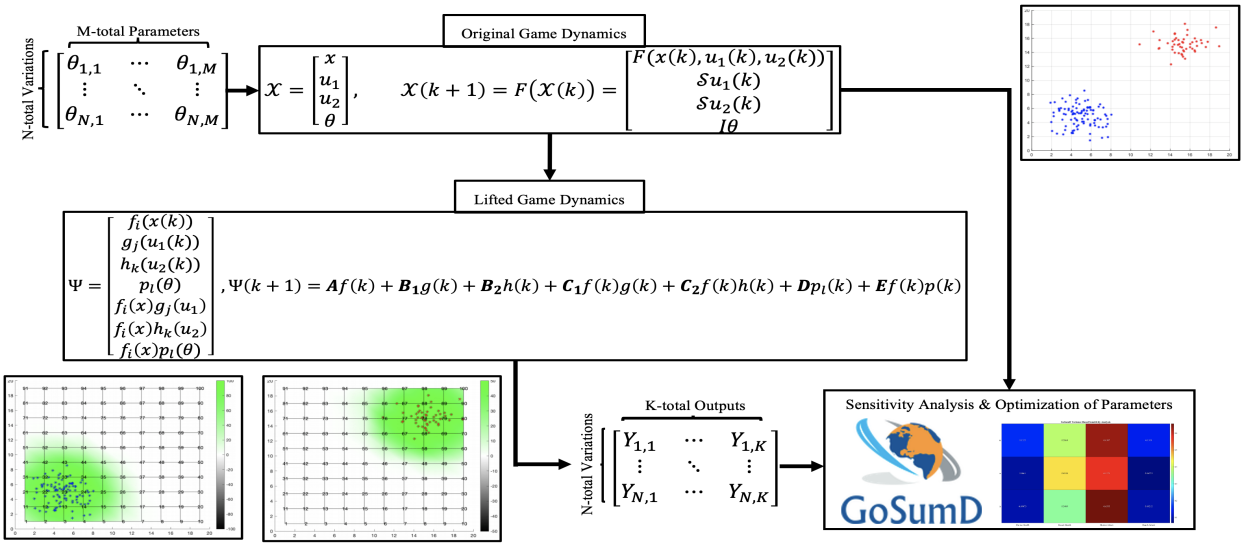

Fig. 1. Flowchart of the Game Balancing Methodology. At the highest level, we generate several parameter variations with which we generate game replay data for training of the Koopman model and sensitivity analysis. The game replay data is used to train the Koopman model on the lifted state space of weighted-densities (Section V-A). Once the Koopman model is trained we can use the model to generate game replay data and produce an equivalent sensitivity analysis via the GoSumD software and compare that the sensitivities of the Koopman model match the sensitivities of the model (Section V). Furthermore, since the game mechanics were included in the lifted state, we can run a minimization of a specified balance criterion with respect to the parameters to obtain optimal parameters that balance the game (Section V-B). In the top right corner of the figure, we demonstrate the original state of a game, consisting of two teams (Red/Blue), and this can be compared with the graphic depicting the lifted state of weighted densities, shown in the bottom left corner of the figure, colored in green.

reduce the complexity of the above mentioned Koopman representations by instead considering the dynamics of the game as a closed-loop system. This is justified when the control inputs $\mathbf{u}_1$ and $\mathbf{u}_2$ are deterministic functions of just the state variable $\mathbf{x}$. This leads to the following closed-loop evolution of the game dynamics

$$\mathbf{x}(k+1) = F(\mathbf{x}(k), \boldsymbol{\theta}) \tag{5}$$

Under the closed-loop dynamics, the observables $\boldsymbol{\psi}$ are only functions of the state and parameter leading to a reduction in the complexity of the Koopman model to the following form

$$\bar{\boldsymbol{\psi}}(\mathbf{x}(k+1)) = \mathbf{M}\boldsymbol{\psi}(\mathbf{x}(k), \boldsymbol{\theta}) \tag{6}$$

Where the observable $\boldsymbol{\psi}$, on the right-hand side of (6), consists of functions of the state, functions of parameters, and products of those functions. The resulting dimension of $\boldsymbol{\psi}$ would be at most $n_x + n_\theta + n_{x\theta}$, where n_x is the number of functions of the state, n_θ is the number of functions of the parameter and $n_{x\theta}$ is the number of products between functions of state and parameter. Lastly, $\bar{\boldsymbol{\psi}}$ appearing in the left-hand side of (6), is a truncation of the full observable which only accounts for the functions of the state and is accordingly of dimension n_x. This is due to the fact that the choice of developing a Koopman operator model on the extended state space is to capture the influence that parameters and control inputs have on the dynamics however, we are ultimately interested in only the evolution of the state and not the parameters or controls. The formulation in equations (3)-(4) are provided to present the full generality and capability of Koopman operator methods for game balancing however, in this work we have considered a Koopman representation given by equation (6) resulting from the closed-loop dynamics of the game dynamics.

III. Learning

A convenient feature of the Koopman model in (6) is that the matrix $\mathbf{M}$ can be estimated from simulation data collected by running a large number of games. Specifically, denoting by θ_i the parameter vector used in the simulation of game $i \in \{1, \ldots, N\}$ and by $\mathbf{x}_i(k) \in \mathbb{R}^{n_x}$ the corresponding state of the game at time $k \in \{1, \ldots, K\}$, we can estimate the matrix $\mathbf{M} \in \mathbb{R}^{n_x \times (n_x + n_\theta + n_{x\theta})}$ using the following least squares problem

$$\min_{\mathbf{M}} \sum_{i=1}^{N} \sum_{k=1}^{K-1} \|\mathbf{M}\boldsymbol{\psi}(\mathbf{x}_i(k), \boldsymbol{\theta}) - \bar{\boldsymbol{\psi}}(\mathbf{x}_i(k+1))\|^2 =$$
$$\min_{\mathbf{M}} \sum_{j=1}^{n_x} \sum_{i=1}^{N} \sum_{k=1}^{K-1} \|m_j\boldsymbol{\psi}(\mathbf{x}_i(k), \boldsymbol{\theta}) - \bar{\boldsymbol{\psi}}_{\boldsymbol{j}}(\mathbf{x}_i(k+1))\|^2,$$

where m_j denotes the jth row of the matrix $\mathbf{M}$ and $\bar{\boldsymbol{\psi}}_j(\mathbf{x}(k+1))$ is the jth entry of the vector $\bar{\boldsymbol{\psi}}(\mathbf{x}(k+1))$. Straightforward algebra can be used to show that the rows m_j of the matrix $\mathbf{M}$ can be obtained independently by solving the following system of linear equations

$$\Big(\sum_{i=1}^{N} \sum_{k=1}^{K-1} \boldsymbol{\psi}(\hat{\mathbf{x}}_i(k))\boldsymbol{\psi}(\hat{\mathbf{x}}_i(k))^{\intercal}\Big) m_j^{\intercal} = \sum_{i=1}^{N} \sum_{k=1}^{K-1} \boldsymbol{\psi}(\hat{\mathbf{x}}_i(k))\bar{\boldsymbol{\psi}}_j(\mathbf{x}_i(k+1)). \tag{7}$$

One should note that the computation needed to construct the matrix and vector in the left- and right-hand sides of equation (7) equation, respectively, scales linearly with the number of games N used to learn data, whereas the memory complexity scales with the size of the matrix $\mathbf{M}$, but not with N. This enables the use of a very large number of simulations at a small computational/memory cost.

IV. Game balancing

Game balancing can be viewed as imposing a desirable behavior for the evolution of the game, such as keeping the "scores" of the two players as similar as possible. Formally, this corresponds to selecting values for the parameter θ to minimize a criterion of the form

$$J(\mathbf{x}(0); \theta) = \sum_{k=k_{\text{init}}}^{k_{\text{final}}} |\text{sc}_1(\mathbf{x}(k)) - \text{sc}_2(\mathbf{x}(k))|^2 \tag{8}$$

where $\text{sc}_1(\mathbf{x}(k))$ and $\text{sc}_2(\mathbf{x}(k))$ denote the score functions of players 1 and 2, respectively, at the states $\mathbf{x}(k)$; and $k_{\text{init}}, k_{\text{init}}+1, \ldots, k_{\text{final}}$ the range of times for which we seek to keep the scores similar.

When using a Koopman representation of the game, minimizing a criterion of the form (8) becomes especially easy if the scores $\text{sc}_1(\mathbf{x}(k))$ and $\text{sc}_2(\mathbf{x}(k))$ are included in the set of observables. In this case, the criterion (8) becomes a quadratic function of the observables, which evolve according to the linear dynamics in (6), leading to a convex quadratic optimization. While many solvers are available to solve such problems, in our work we used the toolbox [21], which provides a convenient interface that can be used to add constraints or additional cost terms that can be used to penalize unsuitable choices of parameters.

V. StarCraft II Minigame

In this work, we have chosen to apply the above-described balancing methodology to the popular real-time strategy game known as StarCraft II (SC2). The game centers around three species known as the Terrans, the Zerg, and the Protoss. While the objective of the game is to simply destroy your opponent in battle, SC2 is known to have vast and complex game mechanics that can involve the collection of resources, the building up of structures, and soldiers among many others. Due to its popularity and complexity, SC2 has been the center of numerous AI research projects which attempt to develop AI players. For example, the DeepMinds division within Google has developed an SC2 minigame platform, known as the DefeatRoaches minigame, for the development and testing of their AI players. Specifically, the minigame consists of a battle between a team of Marines from the Terran race and a team of Roaches from the Zerg race. Due to the availability of pre-trained AI players, we have chosen to apply our methodology to the balancing of the DeepMinds SC2 minigame. The agents used in the generation of the datasets were the pre-trained AI agents developed by Google Deepminds to play the DefeatRoaches minigame. The same agents were used for every parameter combination considered in this study so that the only differences in the outcomes of the game arise from parameter changes and initial conditions. The assumption is made that agents are relatively matched in skill level throughout the range of explored parameter settings. For this SC2 miniGame, we have chosen 4 game parameters Marine Health, Marine Attack, Roach Health, and Roach Attack as the modifiable game mechanics. Health is the number of life units an agent has and Attack is the number of life units an agent can remove from an enemy's health. Additionally, there are two relevant in-game metrics called Score and Killed Value Units (KVU) which measure the state of the game and are calculated by the minigame itself as follows

$$\begin{aligned} \text{Score} &= 10 \times N_{RK} - 1 \times N_{MK} \\ KVU &= 100 \times N_{RK} \end{aligned} \tag{9}$$

where N_{RK} and N_{MK} are the number of Roaches and Marines Killed. Note, the in-game Score metric above in equation (9) is not to be confused the the score functions appearing in the balance criterion (8).

A. Koopman Modeling of StarCraft II

In order to develop the Koopman model, we must first select the type of observables to use for lifting the

dynamics. For this analysis, we use the health-weighted densities of players as functions of the state as well as the product between parameters and the health-weighted densities. The health-weighted densities are obtained by partitioning the game domain into N_{box} boxes, taking a count of the number of Marines and Roaches that fall within each box and scaling the count by the respective health of the agents. We also include the Score and KUV metrics into the lifted state and choose $N_{box} = 100$, resulting in 202 functions of the state. The previously mentioned choices ultimately lead to a model of the type shown in equation (6), where $n_x = 202$, and $n_{x\theta} = 808$. In order to train the Koopman model, we generated a uniform sampling of 601 parameter variations within $\pm 50\%$ of their default values. We then simulated 10 game replays for every parameter resulting in a data set consisting of 6010 game replays. Figure 2 below demonstrates the health-weighted densities of the players.

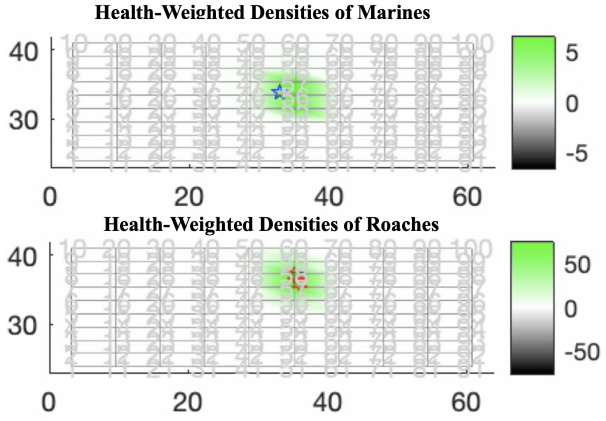

Fig. 2. Plot of the Resulting Health-weighted Densities of Players. The densities are obtained by partitioning the game domain into boxes, taking a count of the corresponding Roaches/Marines within each box and weighing the count according to the health of the agents. Additionally, we also include the product between the health-weighted densities and the game parameters into the lifted state ψ. With the observables in hand we then proceed to compute a finite dimensional representation of the associated Koopman operator which governs their evolution in time.

Prior to balancing the game, we begin by understanding the sensitivity of the Score and KVU (dynamics) to the health and attack parameters (mechanics). This is achieved by utilizing AIMdyn's software known as Global Optimization Sensitivity and Uncertainty in Models and Data (GoSumD). GoSumD is capable of producing a sensitivity analysis from samples of input and output data. The inputs (parameters) and outputs (final Score and final KVU) of the game are fed into GoSumD, and GoSumD utilizes this data to learn an input-to-output model based on Support Vector Regression (SVR) algorithms. Once the SVR model is learned, its derivatives with respect to inputs are utilized for producing a sensitivity analysis to determine which parameters influence which outputs the most or least. In addition to determining which parameters affect the dynamics of the game the most, the sensitivity analysis can also be used to validate the learned Koopman model by comparing the sensitivity of the game data to the sensitivity of the learned Koopman model. In figure 3 we demonstrate a comparison between the sensitivities of the SC2 data and the learned Koopman model. It is clear to see that both outputs are most sensitive to changes in the Marine attack while variations in the other parameters don't seem to affect the game outputs. Interestingly, there is a well-known strategy amongst SC2 players which consists of upgrading the attack of the marine units early in the game in order to obtain a tactical advantage and the sensitivity analysis is in line with this known strategy. Furthermore, the sensitivity of the Koopman model matches the sensitivity of the SC2 data, validating the fidelity of the learned model and its ability to reproduce the game dynamics.

Once the Koopman model is trained and validated we can proceed to develop a balance criterion. For this work we chose a balance criterion according to equation (8) with the specific score functions shown below

$$sc_1(x) := \frac{\sum_M H_M(k)}{\sum_M H_M(0)}, \quad sc_2(x) := \frac{\sum_R H_R(k)}{\sum_R H_R(0)}$$

where $H_M(k)$, $H_R(k)$ is the health of the Marines and Roaches at step k and the scores are computed for $k_{init} = 15$ and $k_{final} = 20$. Essentially, the balance criterion J can be thought of as the two-norm of the difference between the normalized health of the game agents. This balance criterion is a function of the game mechanics and serves to measure how balanced the dynamics are. Thus, we solve a minimization problem to find the set of balanced parameters which produce near-zero values of J. A flowchart of the game balancing methodology can be referenced in figure 1.

B. Balancing Surface

The space of parameters for this SC2 minigame is four-dimensional and it turns out that there is a locust of points inside this four-dimensional space that yield a low value of the balance criterion J. We refer to this locust of balanced points as the balanced surface and attempt to solve the minimization problem along slices of this surface. This is achieved by holding two parameters fixed while minimizing the other parameters. Figure 4 below demonstrates the results for slices along varying

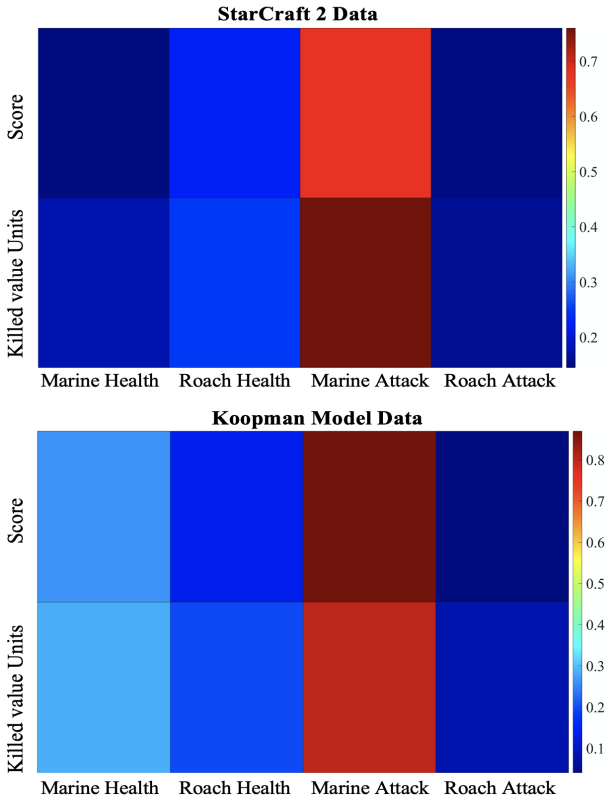

Fig. 3. Comparison of SC2 and Koopman Model Sensitivity Analysis. Each column represents a parameter and each row represents an output. The color scale is such that blue corresponds to low sensitivity and red corresponds to high sensitivity. It is clear to see that both outputs are most sensitive to changes in the Marine attack while variations in the other parameters don't seem to affect the game outputs. Furthermore, the sensitivity of the Koopman model very closely matches the sensitivity of the SC2 data validating the fidelity of the learned model and its ability to reproduce the game dynamics. In order to train the Koopman model, we generated a uniform sampling of 601 parameter variations of health and attack of Roaches and Marines within $\pm 50\%$ of their default values. We then simulated 10 game replays for every parameter resulting in a data set consisting of 6010 game replays. The comparison shown here is between the sensitivity of the learned Koopman model and the training data.

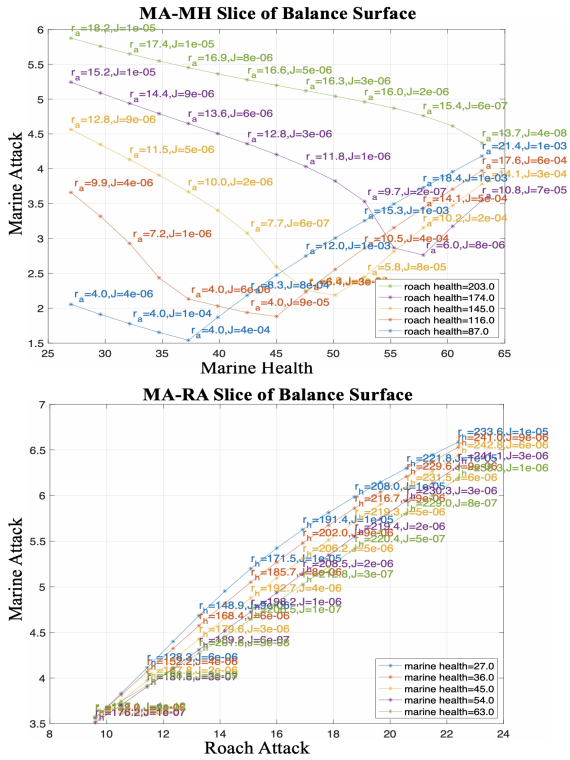

Fig. 4. Two Slices Along the Balanced Surface. Upon solving the minimization problem we find that there are several points for which the balance criterion J has a small value. We plot the locust of such points, referred to as balanced points, and can infer that they seem to form contours of an embedded surface within the four dimensional parameter space. The minimization problem was solved by holding two parameters fixed while varying the others. Thus, we obtain slices along the balanced surface and here we display the Marine attack - Marine Health slice along with the Marine attack - Roach attack slice. It is also interesting to note the apparent curvature of the balanced surface.

Marine attack - Marine Health and varying Marine attack - Roach attack.

The resulting plot of the balanced points appears to be contours along the balanced surface, which itself appears to contain some curvature. The balanced points shown in figure 4 were obtained via the minimization of the balance criterion J subject to the dynamics estimated via the Koopman model. In other words, the balanced points are balanced according to the estimated dynamics. In order to verify that the minimization indeed converged to balanced points, we generated 10 replays of game data for several points within a small neighborhood of the balanced points on the Marine Attack - Roach Attack slice and compute the balance criterion resulting from the SC2 game data. We then utilize that data to verify that the balanced points resulting from the Koopman model are indeed balanced. Figure 5 below demonstrates the validation of the Koopman model's estimation. However, we recall that the data used to train the Koopman model was generated via 601 parameter sets sampled uniformly within $\pm 50\%$ of the default parameter values. Essentially, the training data was sampled within a hyper-rectangle of the entire four-dimensional parameter space. In certain cases, the balanced parameters that resulted from the minimization of J fell outside of this hyper-rectangle and those cases have been colored red in figure 5 below.

To further demonstrate the difference between an unmodified game, specifically [MH=45, RH=140, MA=7, RA=21] and its corresponding balanced game, [MH=45,

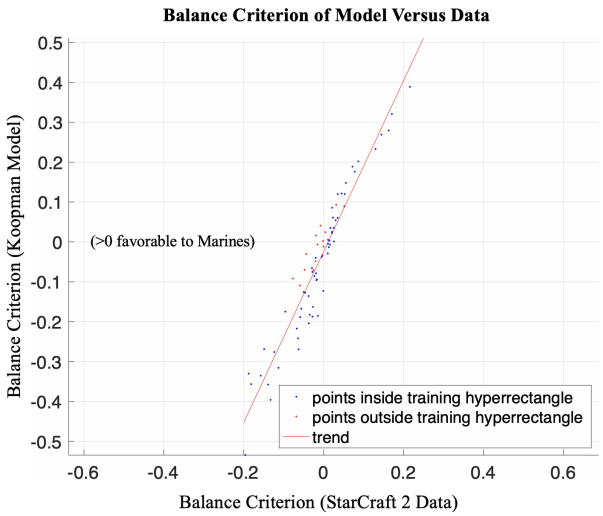

Fig. 5. Validation of the Koopman Based Game Balancing Methodology. Once the minimization problem is solved for the balanced parameters we generate 10 replays of game data for a set of parameters in a small neighborhood of the balanced surface and compute the resulting balance criterion of the games. We then plot the resulting balance criterion from the SC2 minigame against the criterion produced by the Koopman model to verify the fidelity of our balancing approach. If points very close to the computed balanced surface indeed correspond to balanced games we expect to see a linear trend between the Koopman model criterion and the minigame data. It is clear to see that a trend is present amongst a majority of the points with relatively little scattering. Furthermore, we recall that the data used to train the Koopman model was generated via 601 parameter sets sampled uniformly within $\pm 50\%$ of the default parameter values. In certain cases, the balanced parameters that resulted from the minimization of J fell outside of the range of the training data and those cases have been colored red.

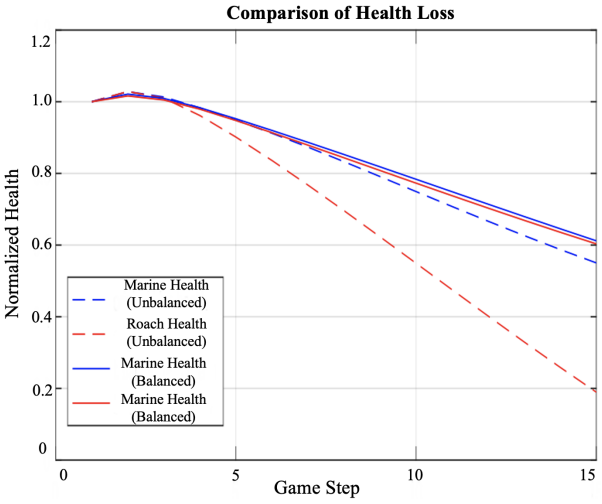

Fig. 6. Comparison of the Normalized Health Curves for an Unmodified Versus Balanced Game. The dashed curves correspond to the unmodified game and clearly demonstrate how the health of the Roaches decay much faster than the Marine's health. The solid curves correspond to the balanced fame and demonstrates how the slopes of the curves are now nearly equal.

RH=140, MA=4.39, RA=17.23] we plot the time series of the normalized health's of the agents, produced by the Koopman model, and compare the slopes of the curves. The dashed curves in figure 6 clearly demonstrate how, in the unmodified game, the health of the Roaches decay much faster than the Marine's health. The solid curves demonstrate the revised evolution of the health after balancing the game and the slopes of the curves are now nearly equal.

Lastly, we compare the game statistics, computed over the 10 replays of the balanced and unmodified parameters, below in table I. It is clear to see that there was an apparent imbalance against the roaches and that the unmodified games ended within roughly 20 game steps. Upon balancing, the final health of the players is indeed statistically near zero, as expected since the balance criterion was based on the health of the players. Interestingly, the balanced games now last, roughly three-times longer than the unmodified games. Lastly, the statistics show that the balanced game does not produce a $50/50$ probability of winning and the reason for this is that StartCraft is a fully deterministic game. Hence, if you start from the same initial condition the same team will always win, even if both teams have near-zero health at the end of the game. The reason why Marines win 2 games out of ten, is because 2 of the initial conditions slightly favor the marines and the other 8 slightly favor the roaches. Overall, in a fully deterministic game, it is problematic to associate balance with the probability of a win, that is why we chose to balance according to the health rather than the probability of winning.

	Original	Balanced
% Marines Win	100%	20%
Game Duration	$14.5 \pm .7$ [Steps]	65 ± 8 [Steps]
Final Marine Health	$55\% \pm 4$	$2\% \pm 4$
Final Roach Health	$3\% \pm 2$	$16\% \pm 12$

TABLE I
BALANCED VERSUS UNBALANCED GAME STATISTICS.

VI. CONCLUSIONS

In this work, we have presented an automated game balancing framework that relies on a novel modeling of the relation between game dynamics (balance) and the game mechanics (parameters). The game dynamics estimation is based on Koopman operator techniques which have previously never been applied to this domain. The

framework leverages high-performance nonlinear programming solvers to optimize a user-specified balancing criterion and the effectiveness of the framework is validated by balancing a StarCraft II minigame. Thus far, our results demonstrate the ability of the framework's ability to learn the game dynamics from a limited set of training data.

Future works lie in scaling the framework to a more complex SC2 minigame and expanding the dimension of the parameter space considered. Furthermore, there is interest in quantifying the effects that the initial conditions, both in the game space and the lifted space, can have on a balanced game. Exploring the sensitivity of game balance to different initial conditions can yield further insight into the dynamics of a game.

REFERENCES

[1] M. Newheiser, "Playing fair: A look at competition in gaming," Mar 2009.

[2] M. Preuss, T. Pfeiffer, V. Volz, and N. Pflanzl, "Integrated balancing of an rts game: Case study and toolbox refinement," in *2018 IEEE Conf. on Comp. Intell. and Games*, 2018, pp. 1–8.

[3] M. Beyer, A. Agureikin, A. Anokhin, C. Laenger, F. Nolte, J. Winterberg, M. Renka, M. Rieger, N. Pflanzl, M. Preuss, and V. Volz, "An integrated process for game balancing," in *2016 IEEE Conf. on Comp. Intell. and Games*, 2016, pp. 1–8.

[4] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *2008 IEEE Symp. on Comp. Intell. and Games*, 2008, pp. 87–94.

[5] G. Bosc, P. Tan, J. Boulicaut, C. Raïssi, and M. Kaytoue, "A pattern mining approach to study strategy balance in rts games," *IEEE Trans. on Comp. Intell. and AI in Games*, vol. 9, no. 2, pp. 123–132, 2017.

[6] S. Bangay and O. Makin, "Generating an attribute space for analyzing balance in single unit rts game combat," in *2014 IEEE Conf. on Comp. Intell and Games*, 2014, pp. 1–8.

[7] R. Leigh, J. Schonfeld, and S. J. Louis, "Using coevolution to understand and validate game balance in continuous games," in *Proc. of the 10th Ann. Conf. on Genetic and Evol. Comp.* Association for Computing Machinery, 2008, p. 1563–1570.

[8] B. O. Koopman, "Hamiltonian systems and transformation in hilbert space," *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, 1931.

[9] B. O. Koopman and J. V. Neumann, "Dynamical systems of continuous spectra," *Proceedings of the National Academy of Sciences of the United States of America, 18(3):255*, 1932.

[10] I. Mezić and A. Banaszuk, "Comparison of systems with complex behavior," *Physica D: Nonlinear Phenomena*, vol. 197, no. 1, pp. 101 – 133, 2004.

[11] I. Mezić, "Spectral properties of dynamical systems, model reduction and decompositions," *Nonlinear Dynamics*, vol. 41, pp. 309–325, Aug 2005.

[12] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, p. 5–28, 2010.

[13] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis, "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, 2017.

[14] Y. Susuki and I. Mezić, "A prony approximation of koopman mode decomposition," in *2015 54th IEEE Conference on Decision and Control (CDC)*, Dec 2015, pp. 7022–7027.

[15] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Scence*, vol. 25(6), pp. 1307–1346, 2015.

[16] D. Giannakis, "Data-driven spectral decomposition and forecasting of ergodic dynamical systems," *Applied and Computational Harmonic Analysis*, vol. 47, no. 2, pp. 338 – 396, 2019.

[17] C. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. Henningson, "Spectral analysis of nonlinear flows," *Journal of Fluid Mechanics*, vol. 641, pp. 115–127, 2009.

[18] H. Arbabi and I. Mezić, "Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator," *SIAM Journal on Applied Dynamical Systems*, vol. 16, no. 4, pp. 2096–2126, 2017.

[19] H. Arbabi and I. Mezić, "Study of dynamics in post-transient flows using koopman mode decomposition," *Physical Review Fluids*, vol. 2, Dec 2017.

[20] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149 – 160, 2018.

[21] J. P. Hespanha, "TensCalc — A toolbox to generate fast code to solve nonlinear constrained minimizations and compute Nash equilibria," University of California, Santa Barbara, Santa Barbara, Tech.

Rep., June 2017, available at http://www.ece.ucsb.edu/~hespanha/techrep.html.